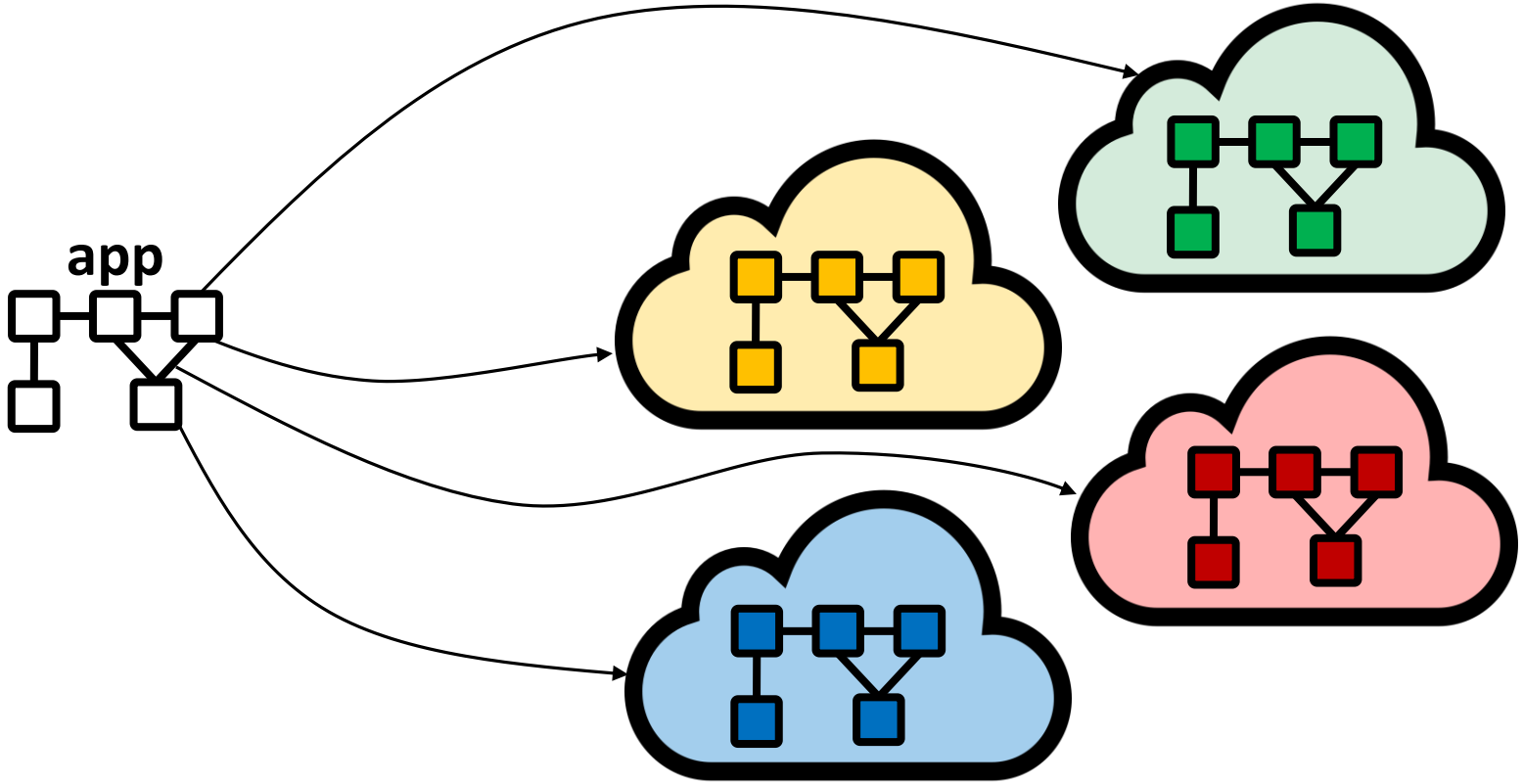
Modelling, analysing and reusing composite cloud applications

Jacopo Soldani
Dipartimento di Informatica, Università di Pisa

SummerSOC 2018, 25th of June 2018

Mail: **soldani@di.unipi.it** Web: **http://pages.di.unipi.it/soldani**

Context



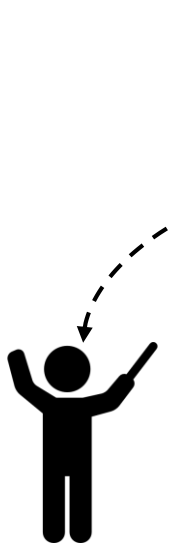
Challenge[1]:

- » Flexibly manage complex **composite applications**
- » over heterogeneous **cloud** platforms.

[1] F. Leymann. **Cloud computing**. it—Information Technology, 53(4):163–164, 2011.

Two major issues[1,2]

Flexibly manage complex **composite applications** over heterogeneous **cloud** platforms.

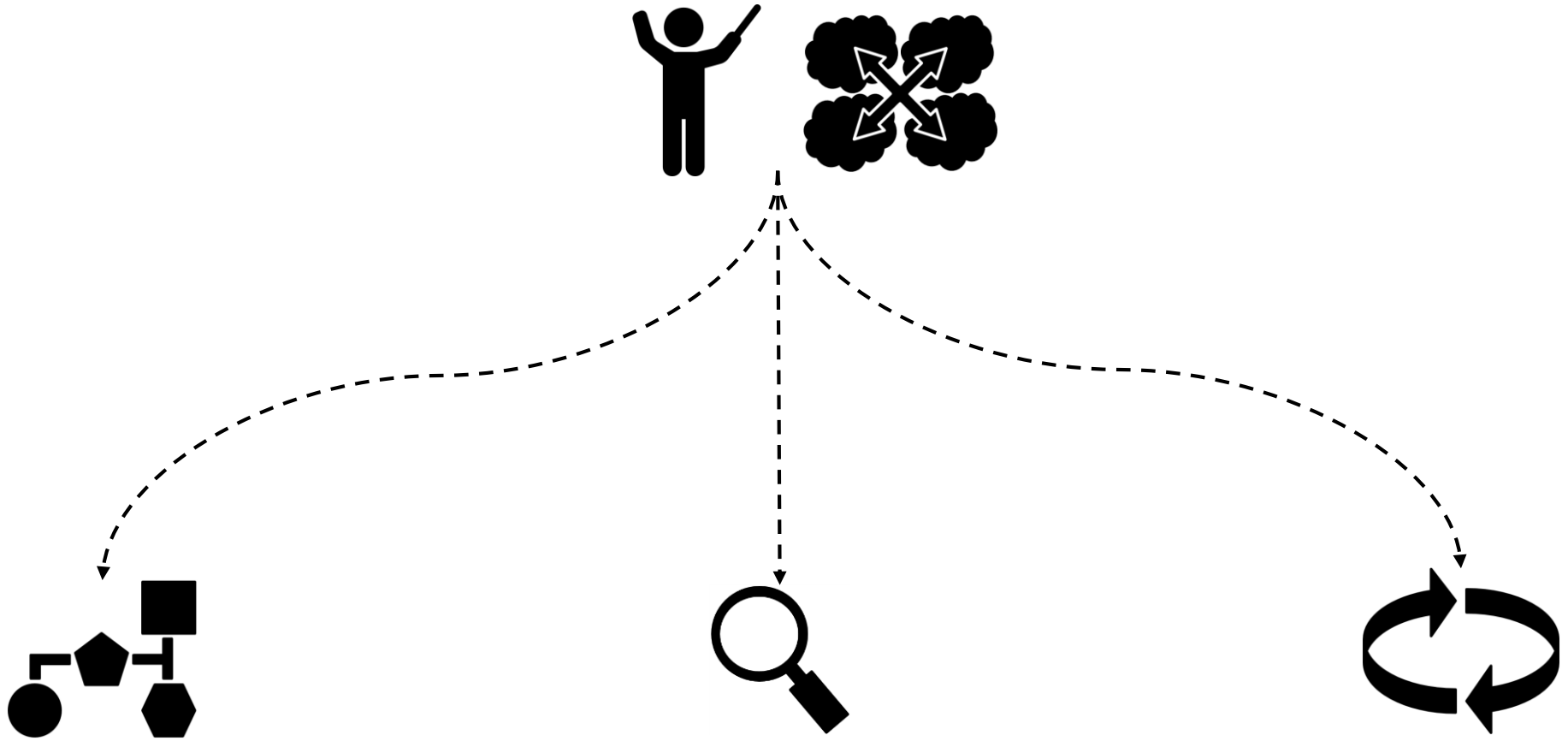


Automate the **management** of composite cloud applications.

Support a **vendor-agnostic design** of composite cloud applications.

[1] T. Binz et al. **TOSCA: Portable automated deployment and management of cloud applications**. Advanced Web Services, pp. 527-549, Springer, 2014.
[2] R. Di Cosmo et al. **Aeolus: A component model for the cloud**. Information and Computation, 239(0):100 –121, 2014.

Research objectives

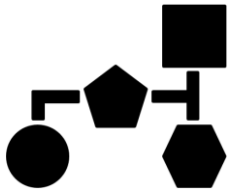


Modelling composite cloud applications.

Analysing composite cloud applications.

Reusing composite cloud applications.

From objectives to research contributions

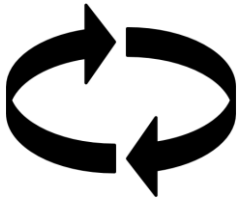


Modelling composite cloud applications.

Compositional, **fault-aware** modelling for the **management behaviour** of applications.

Analysing composite cloud applications.

Techniques for **checking** and **planning** the **management** of applications.

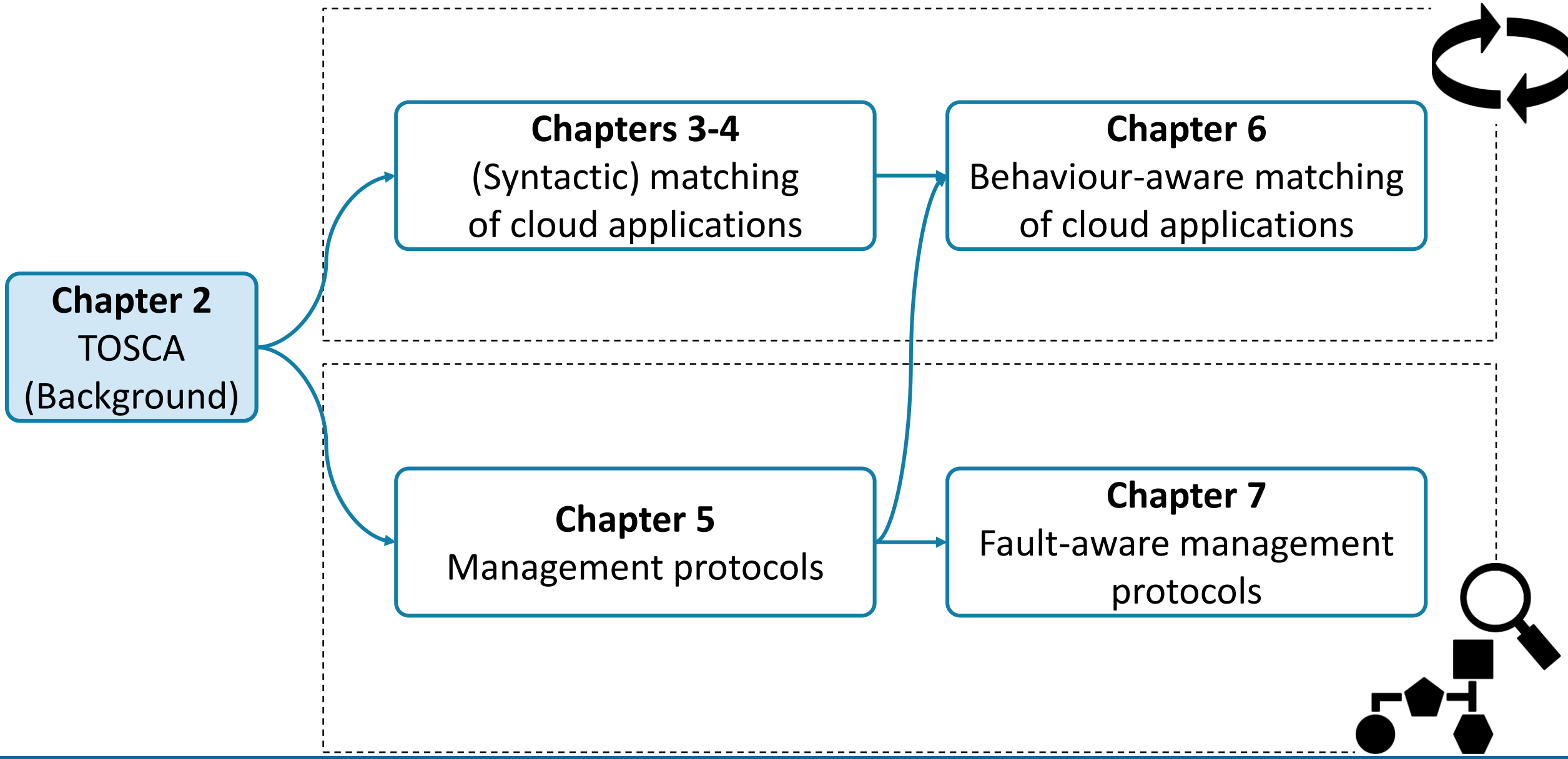


Reusing composite cloud applications.

Techniques for **matching** and **adapting** existing applications.

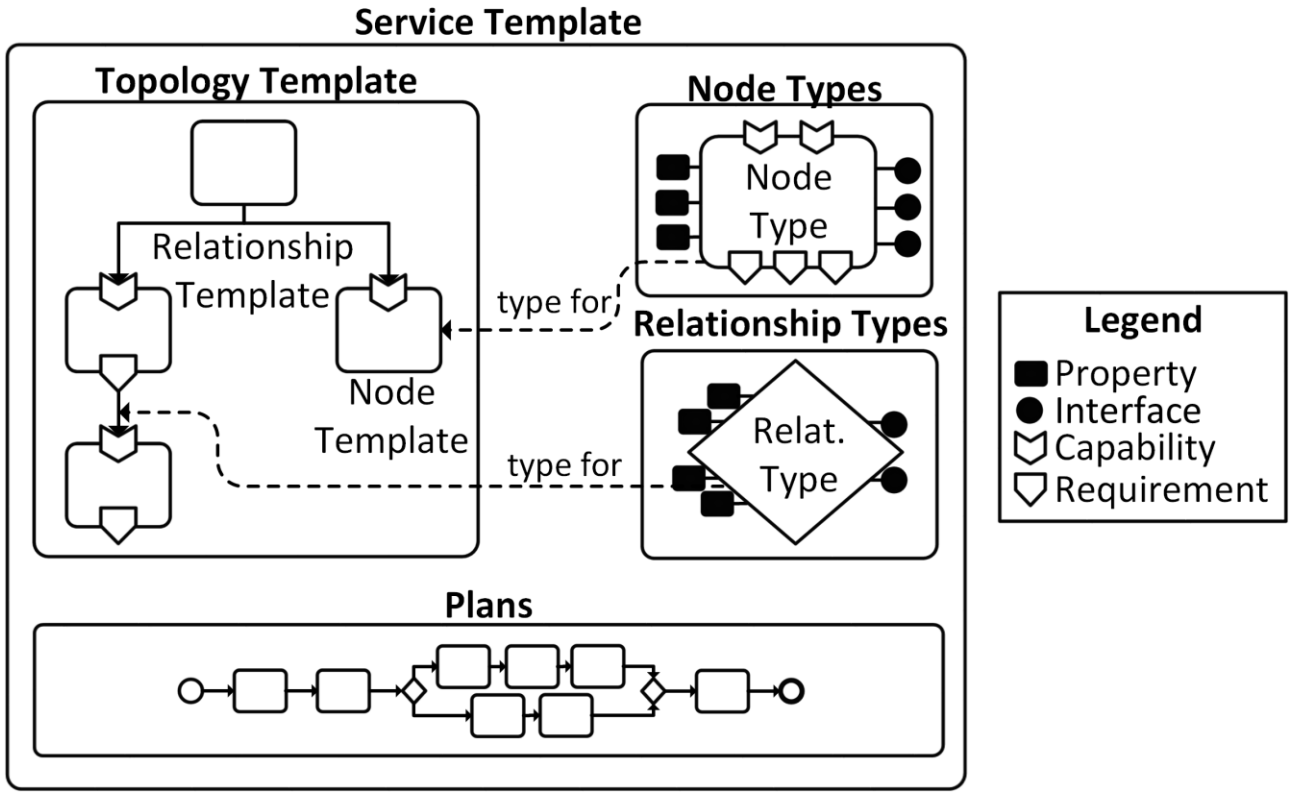
Roadmap

https://tinyurl.com/soldani-thesis

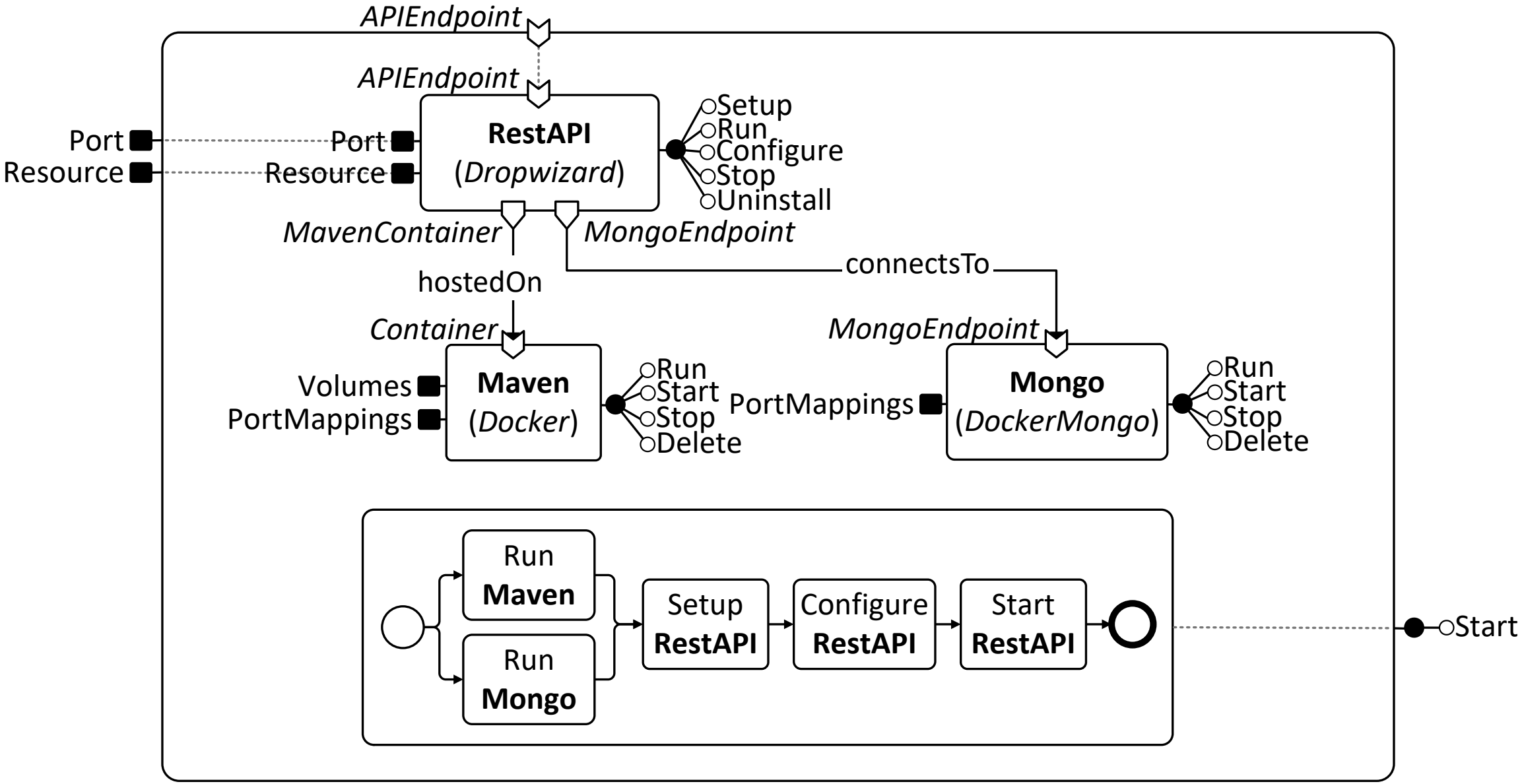


TOSCA (*Topology and Orchestration Specification for Cloud Applications*)

- OASIS **standard**
- **Goals**:
  1. Create portable cloud applications.
  2. Automate application management.

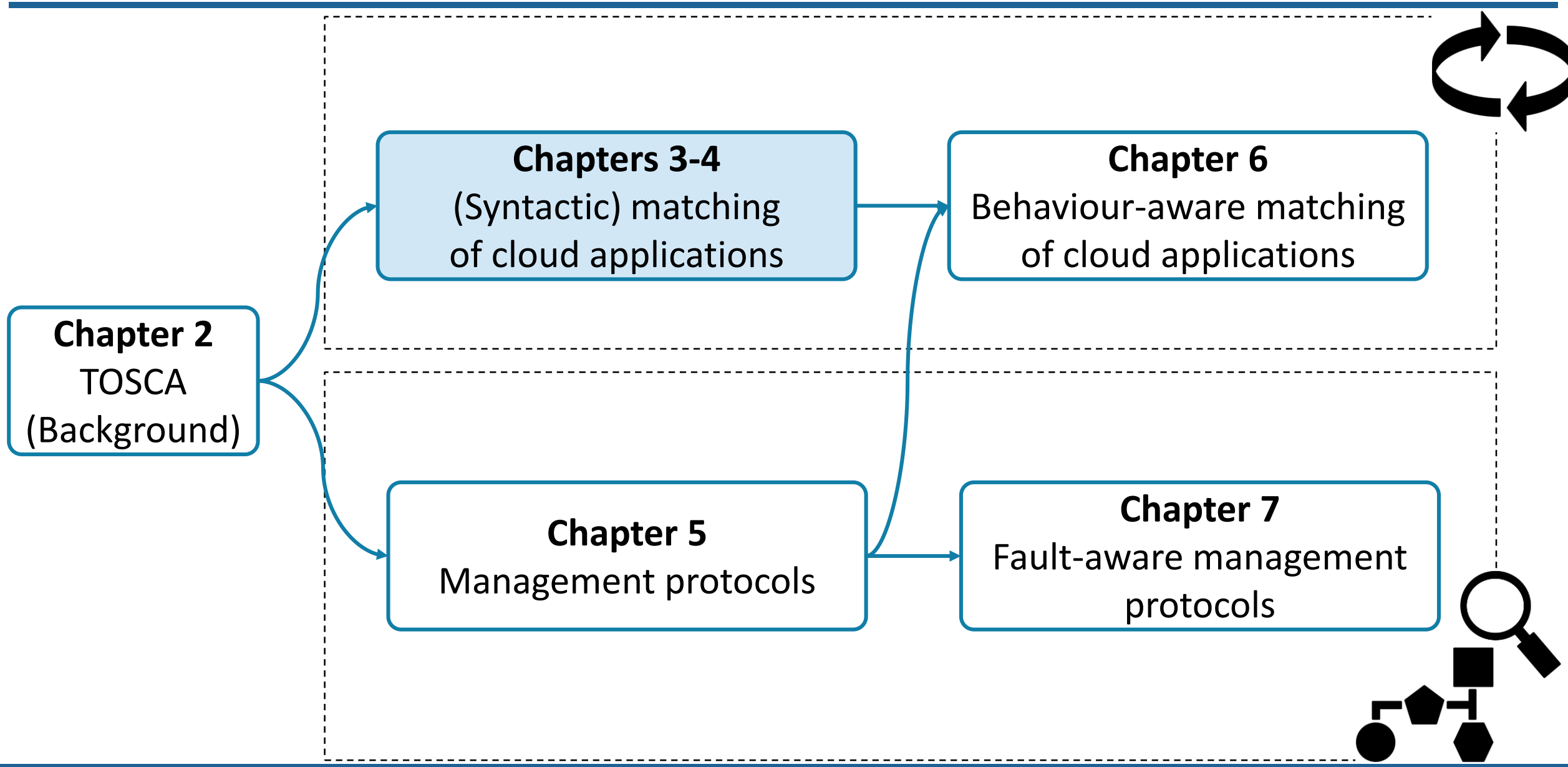


A toy example



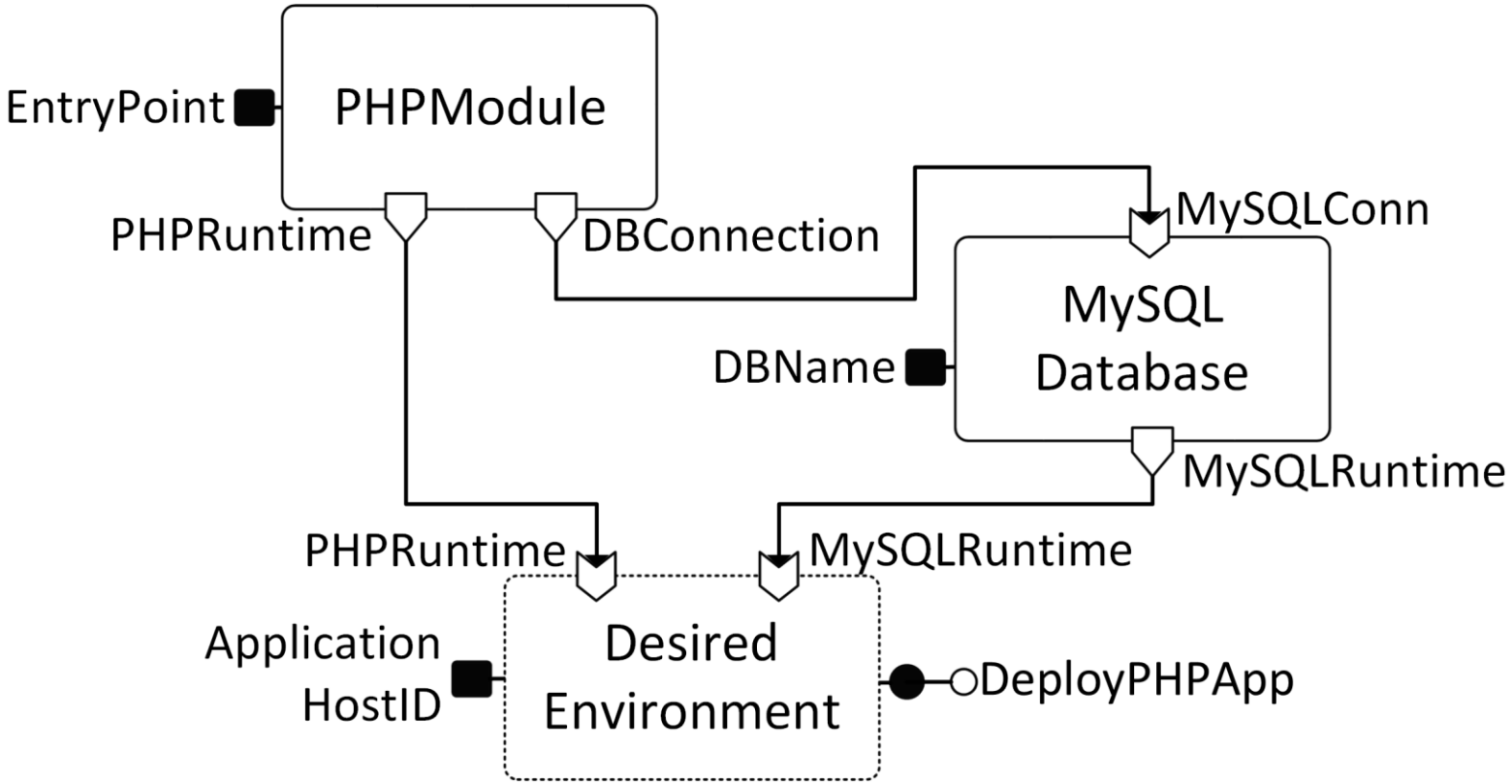
Roadmap

https://tinyurl.com/soldani-thesis



A running example

Objective: Deploy/manage a web application on a cloud.

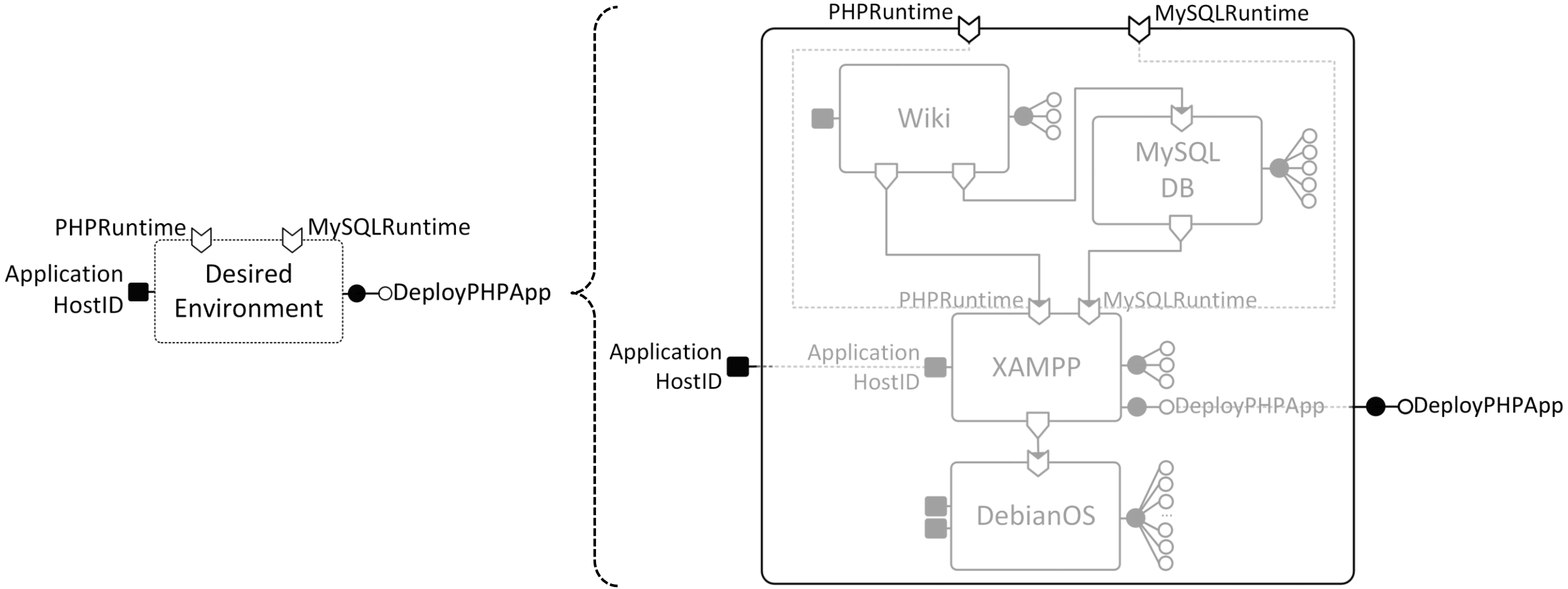


Approach:

1. Abstractly describe the desired hosting environment (*DesiredEnvironment*).
2. Match and adapt existing applications to actually implement *DesiredEnvironment*.

Substitutability of TOSCA applications

Existing TOSCA **applications can be reused** to actually implement desired components[1].



No additional information on how to match nodes/applications is given.

[1] OASIS. **Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer**. 2013.

Four (formal) notions of matching

Exact matching (≡)

extended in — *applications that **"offer more"** and **"require less"***

Plug-in matching (≃) **+ adaptation** methodology

extended in — ***ignore naming differences***

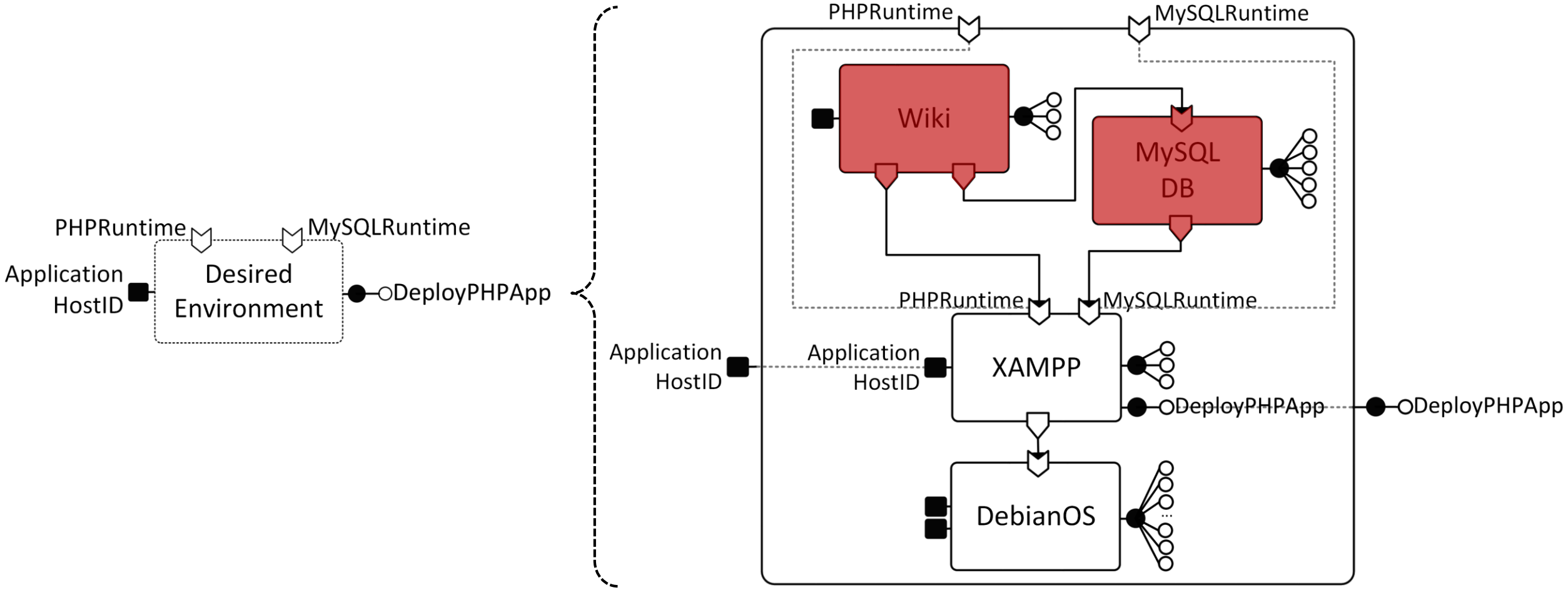
Renaming-based matching (∼) **+ adaptation** methodology

extended in — *search missing features **inside of available applications' topologies***

White-box matching (□) **+ adaptation** methodology

A limitation of the proposed matching notions

All notions of matching (≡,≃,∼,□) permit reusing applications **only in their entirety**.



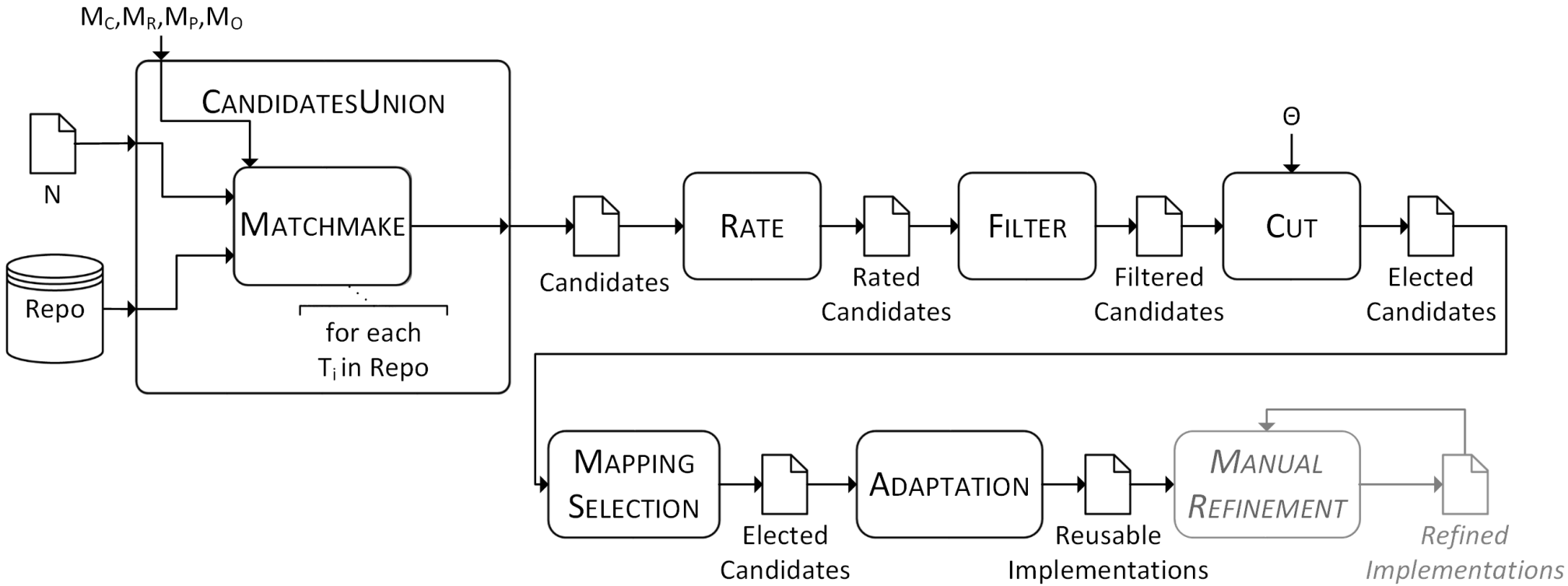
This would potentially **waste resources** to deploy **unnecessary software**.

[1] OASIS. **Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer**. 2013.

Reusing *fragments of* TOSCA applications

TOSCAMART (*TOSCA-based Method for Adapting and Reusing application Topologies*)

» Reuse only the necessary fragments of application topologies.



Properties of TOSCAMART

1. TOSCAMART always **terminates**.

2. TOSCAMART is **sound**.

3. The **time complexity** of TOSCAMART is

$$T(TOSCAMART) = O(rt)$$

where

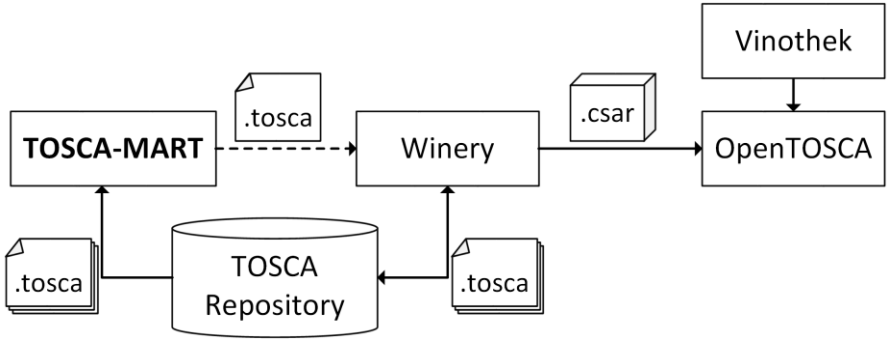
- r is the size of the repository, and
- t is the maximum amount of features available in an application.

Implementation

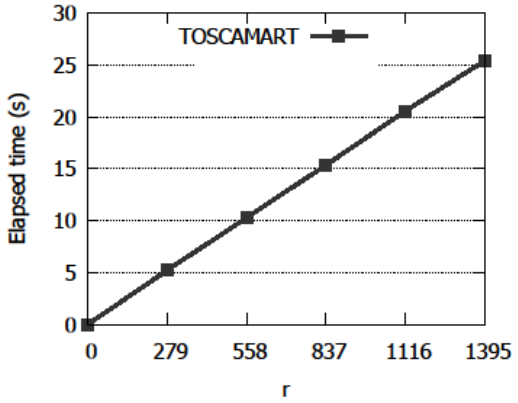
We implemented a **prototype** of TOSCAMART.

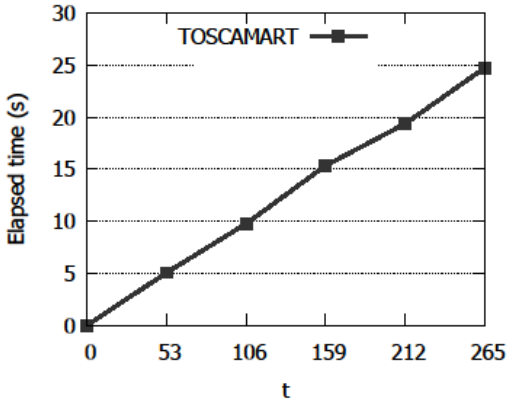
» **Open-source**[1]

» Fully-compatible with the **OpenTOSCA** open-source ecosystem.



» Expected **time performances**

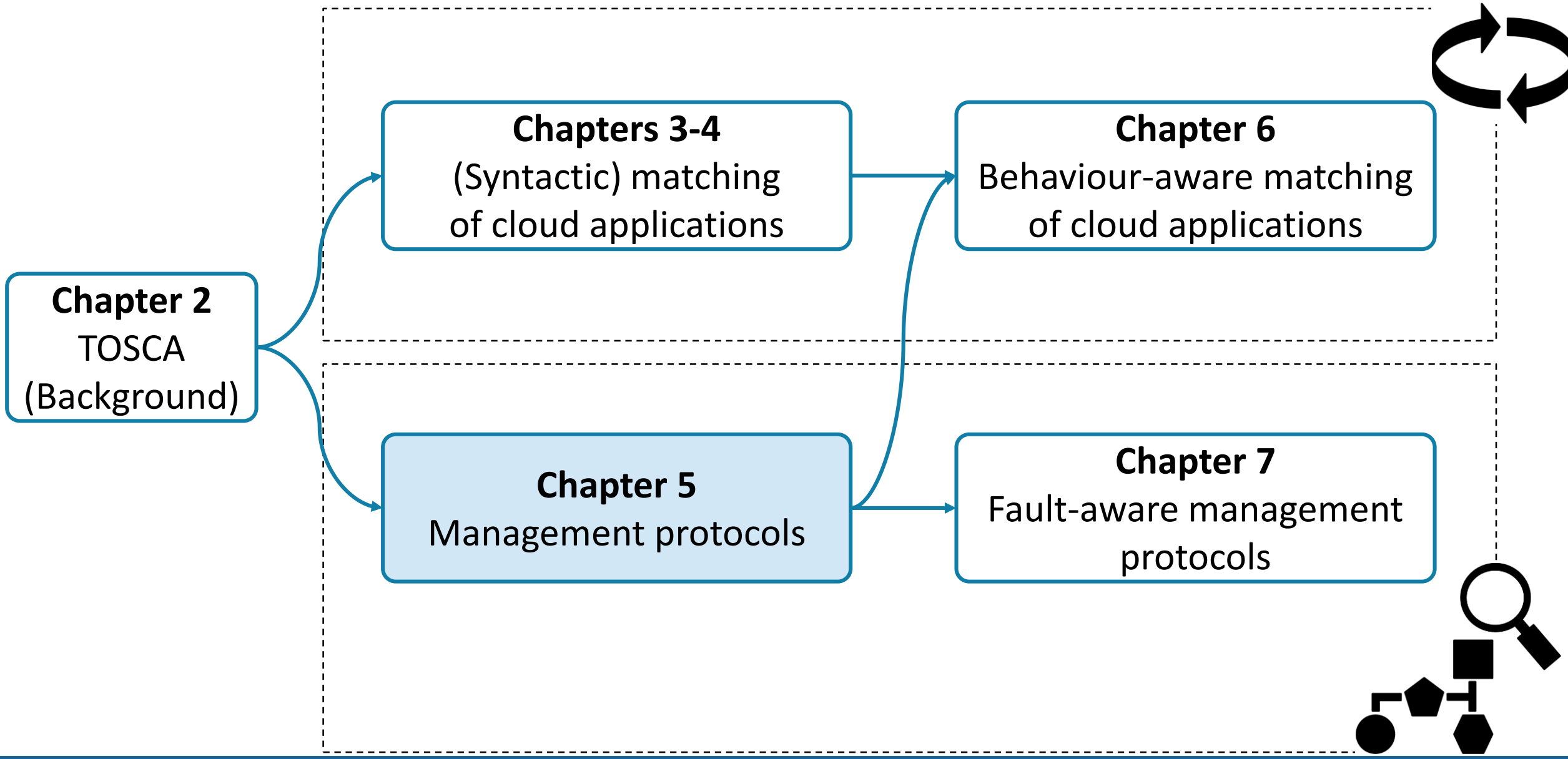




[1] https://github.com/jacopogiallo/TOSCA-MART

Roadmap

https://tinyurl.com/soldani-thesis

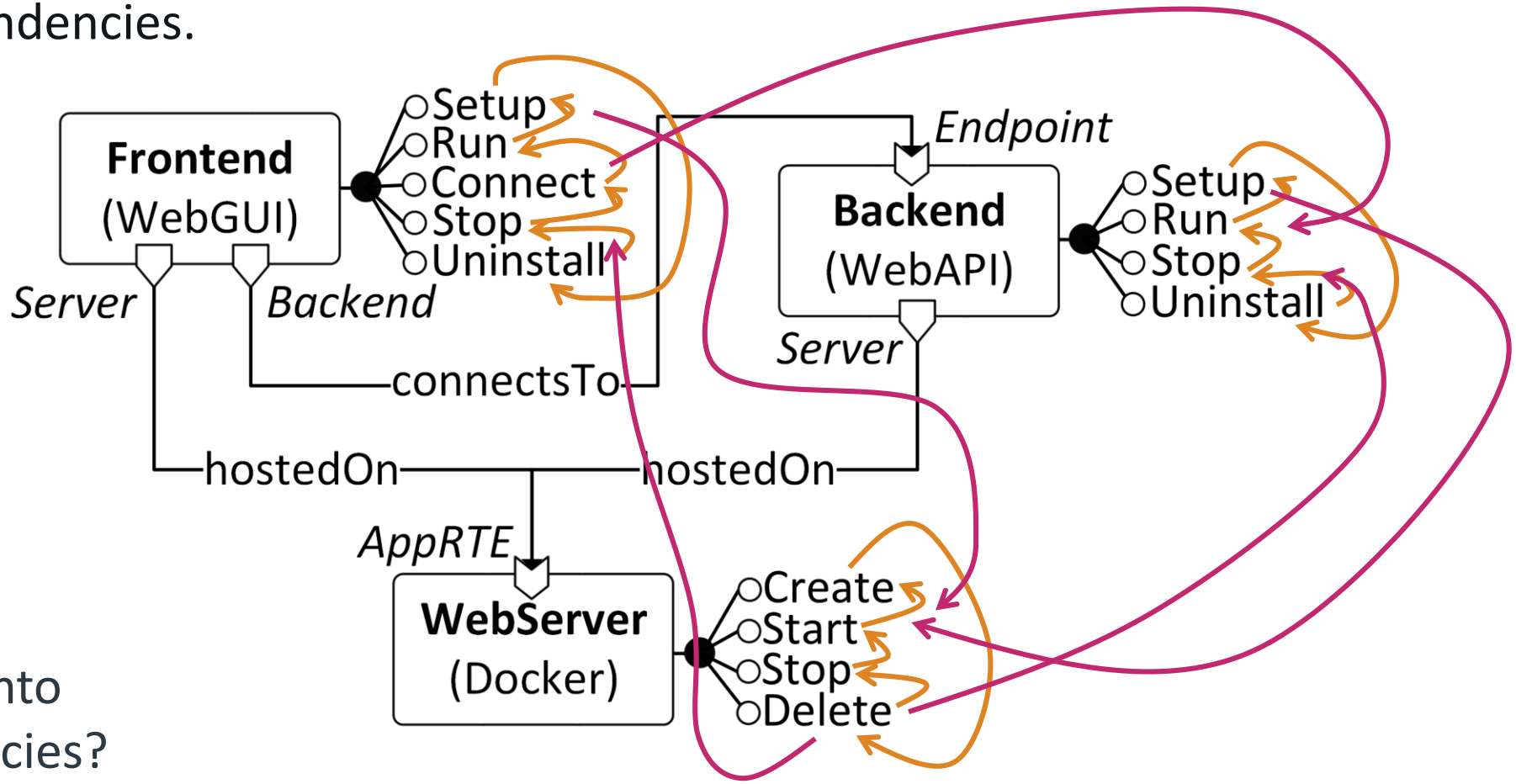


Motivations

Analyse/automate the **management** of composite cloud applications.

» **Intra-component** dependencies.

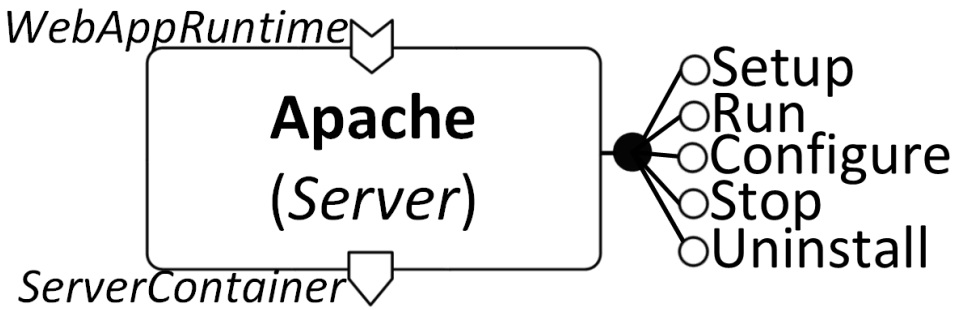
» **Inter-component** dependencies.



How to **easily** take into account **all** dependencies?

Management protocols

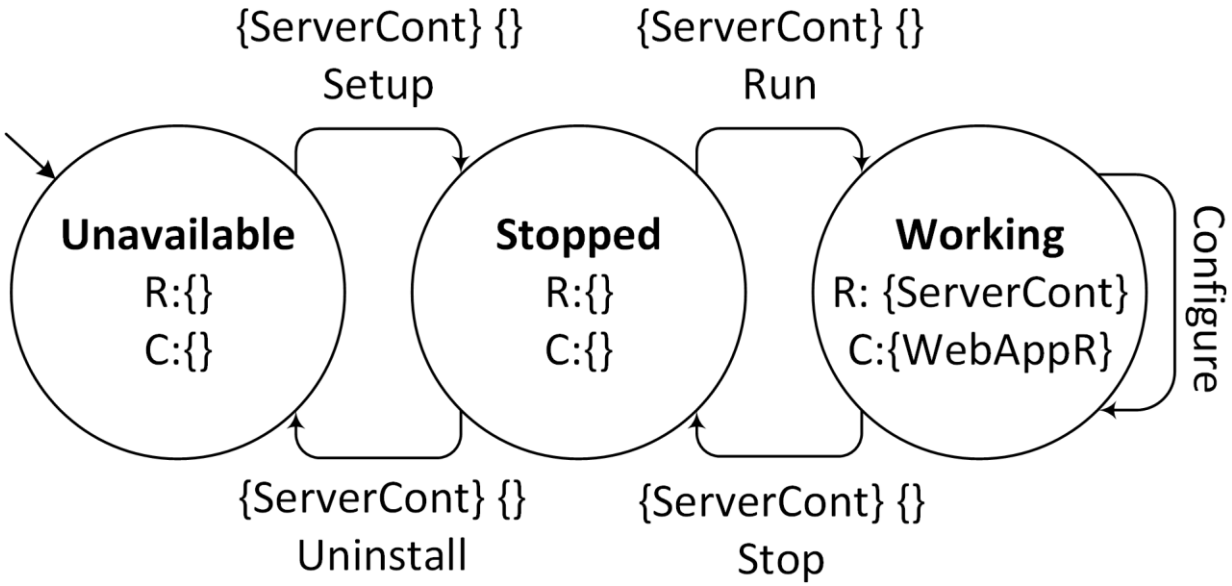
The **management protocol** of a component is a FSM[1].



» Transitions model **intra-component** dependencies.

» Conditions on requirements/capabilities capture **inter-component** dependencies:

- **reqs needed** and **caps offered** in a **state**.
- **reqs needed** to execute a **transition**, and **caps preserved** during its execution.



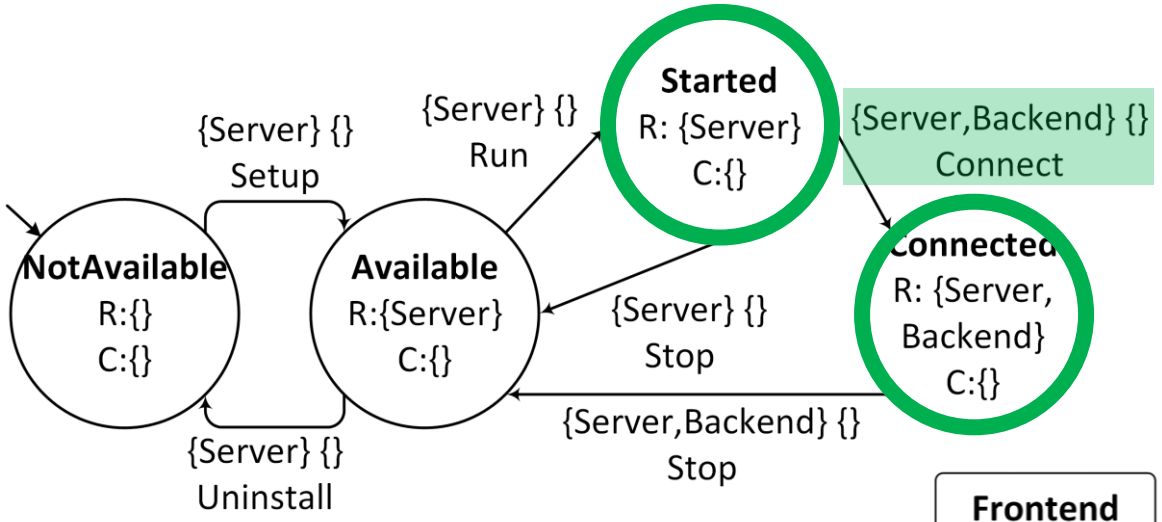
[1] Notions of **well-formedness** and **determinism** of management protocols are defined and can be automatically checked.

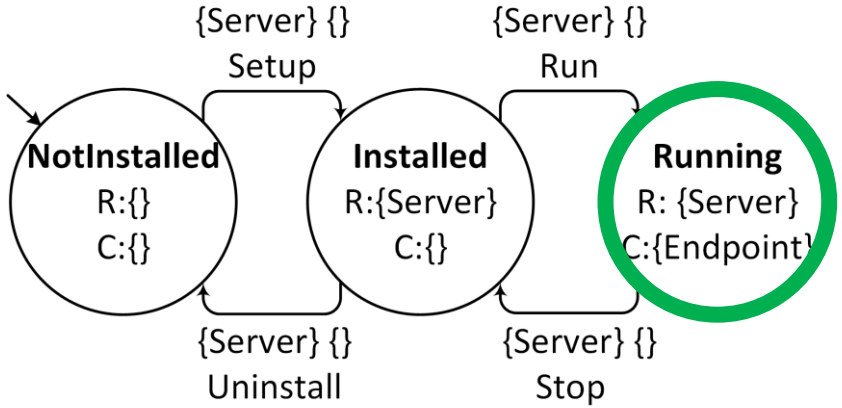
Reasoning with composite applications

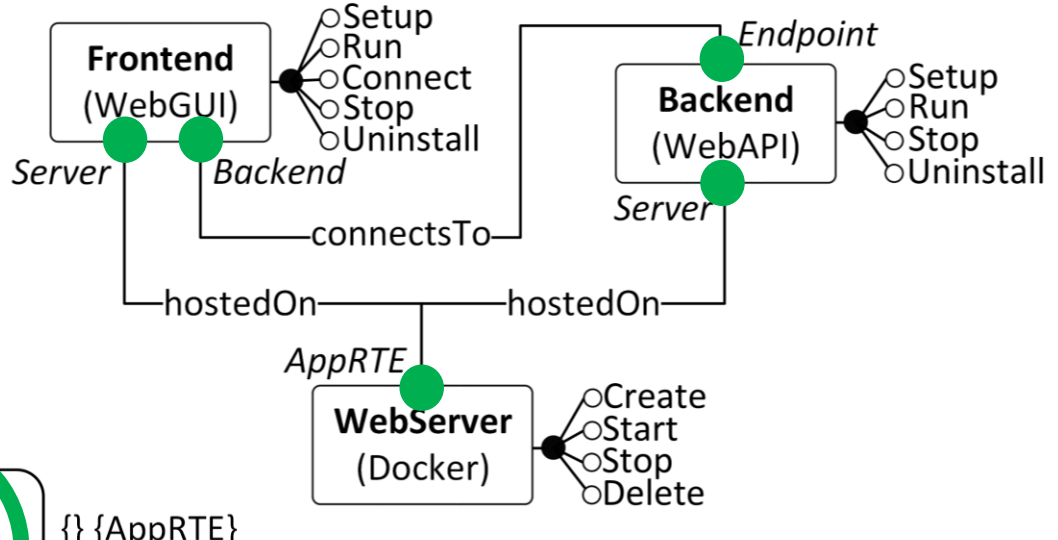
The **management behaviour** of a composite application is derived by composing the management protocols of its components.

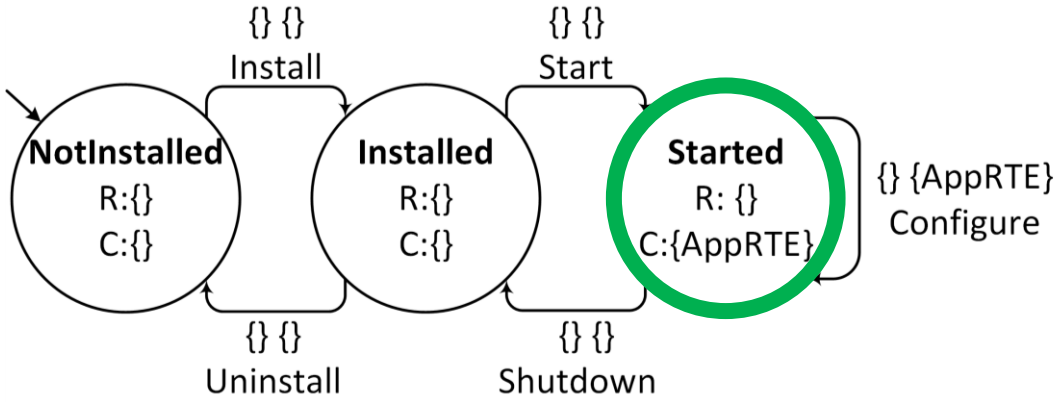
- A **global state** G is a set containing the current state of each component.
- A global state G is **consistent** iff all the requirements assumed in G are satisfied.
- An **operation can be executed** in G iff all the requirements it needs are satisfied in G.

Example – Consistent global state & operation execution









Analysing the management of applications

Validity of plans

- A **sequence** of management operations $o_1 o_2 \ldots on$ is **valid** from a global state G_0 iff $G_0 \xrightarrow{o_1} G_1 \xrightarrow{o_2} G_2 \xrightarrow{o_3} \ldots \xrightarrow{o_n} G_n$ and each G_i is consistent.
- A **workflow plan** is **valid** from a global state G_0 iff all its sequential traces are valid from G_0.

Effects of (valid) plans

- The **effects** of a plan (on states, requirements, capabilities) can be directly determined from global states.
- A valid **plan** is also **deterministic** if all its sequential traces reach the same global state.

Finding plans (achieving desired goals)

- The problem can be solved with a visit of the graph of reachable global states.

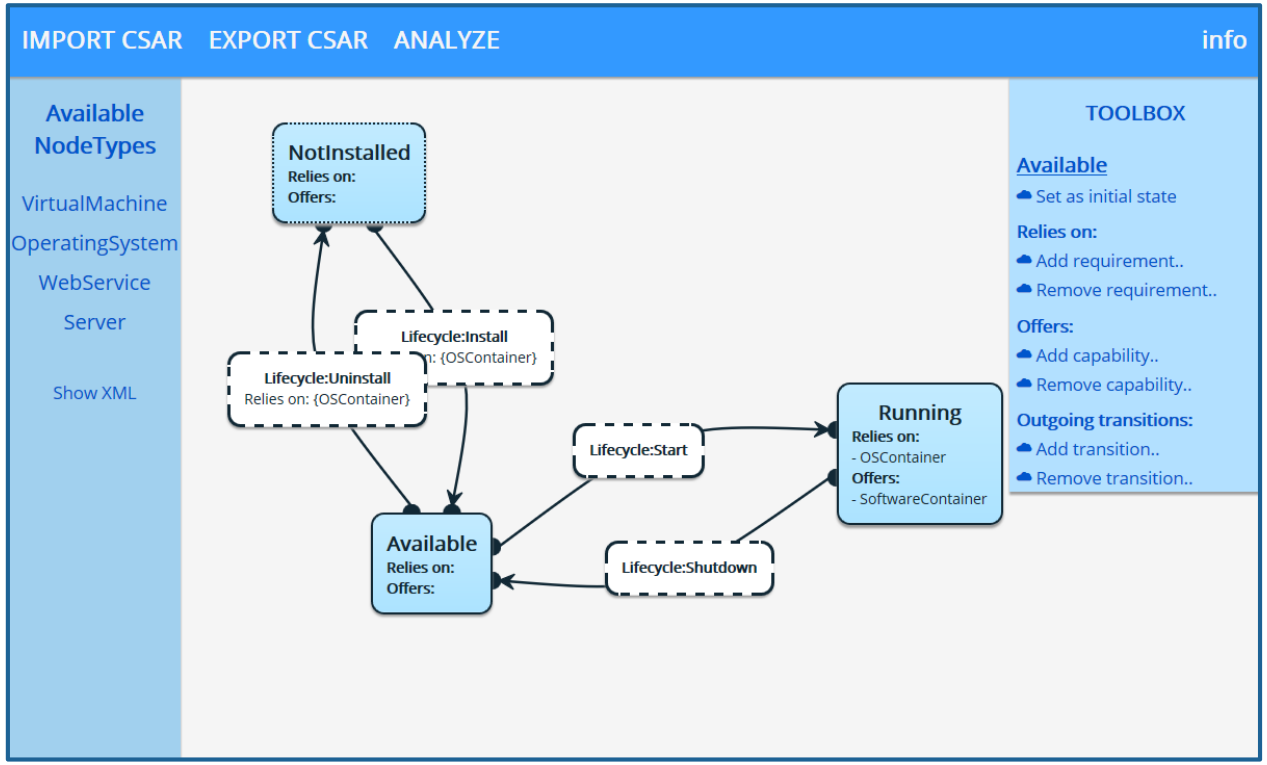
…

Implementation

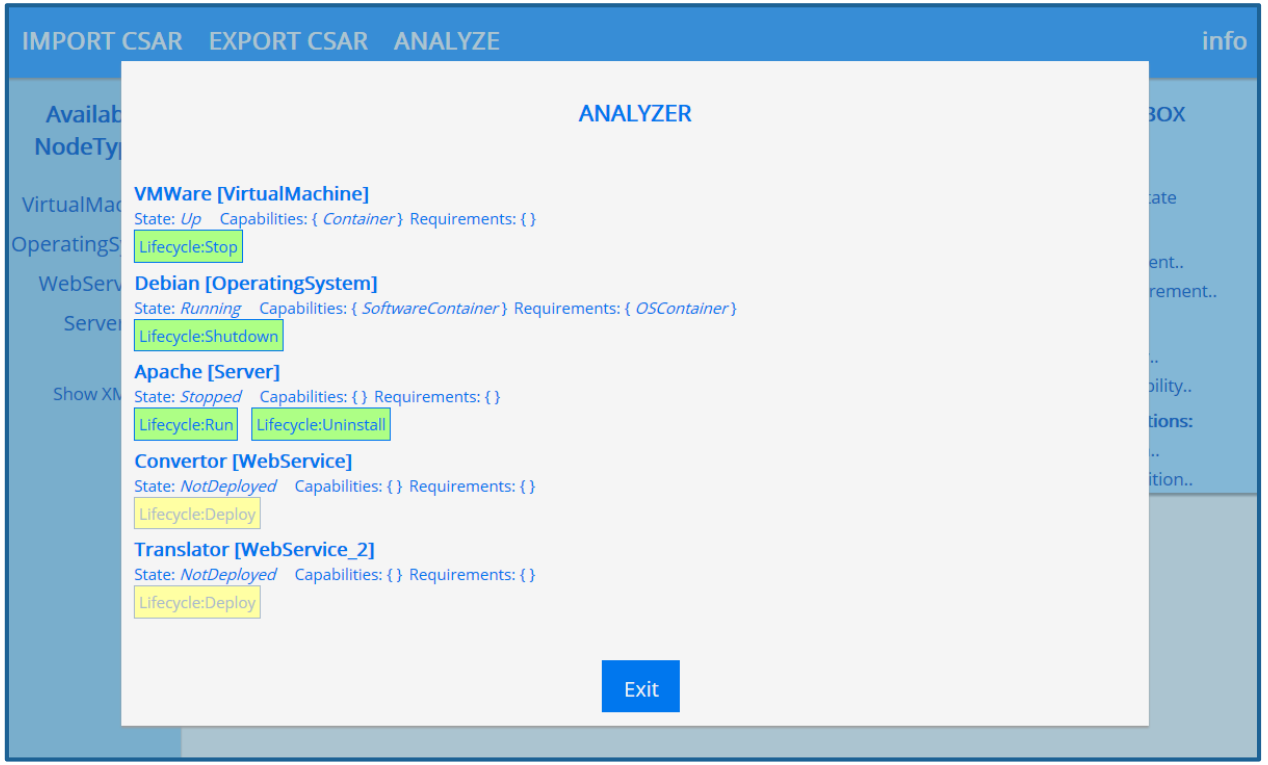
Barrel[1]

» Web-based **editor/analyser** of management protocols in TOSCA applications.

» **Open-source** and compatible with the OpenTOSCA ecosystem.



edit



analyse

[1] http://ranma42.github.io/MProt.

Case study

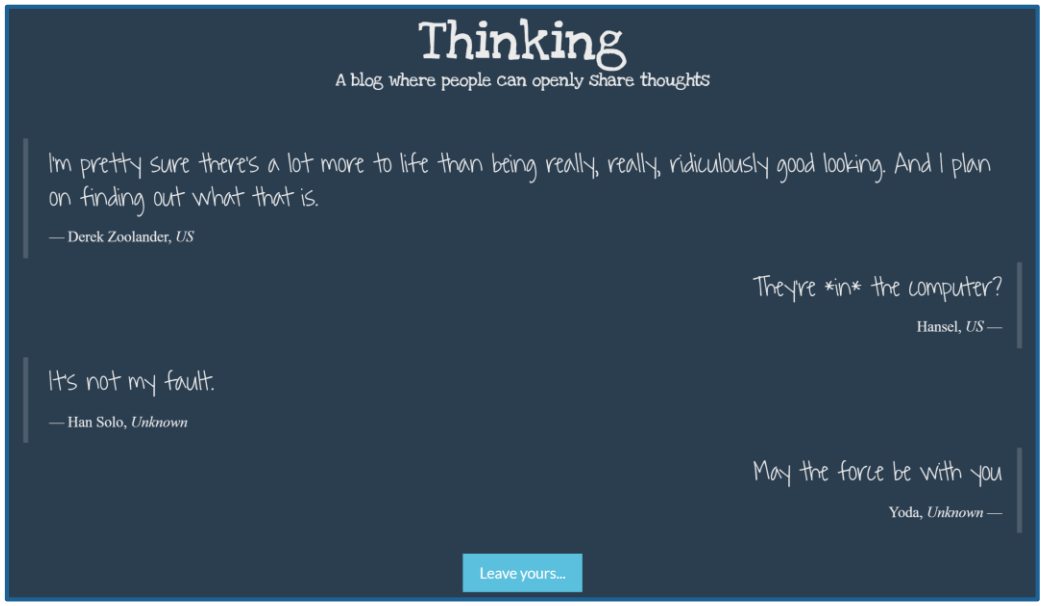
Thinking

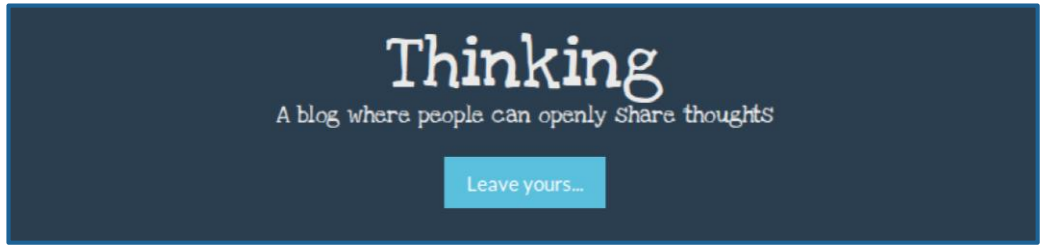
» Real application, made by three components
- **GUI** (deployed on a **NodeJS** Docker cont.)
- **REST API** (deployed on a **Maven** Docker cont.)
- **Mongo** database (running as a Docker cont.)

» **Validation** and **test** of existing deployment plans
- Valid plans effectively deploy application.
- Non-valid plans resulted in crashes/exceptions.

» **Planning**
- Valid plan to undeploy GUI and REST API (only)
- Effectively resulted in undeploying them.

```
jacopo@yellow:~$ docker ps -a
CONTAINER ID   IMAGE  COMMAND                  CREATED         STATUS         PORTS
f385a566c619   mongo  "/entrypoint.sh mongo"  17 minutes ago Up 17 minutes 27017/tcp
jacopo@yellow:~$
```

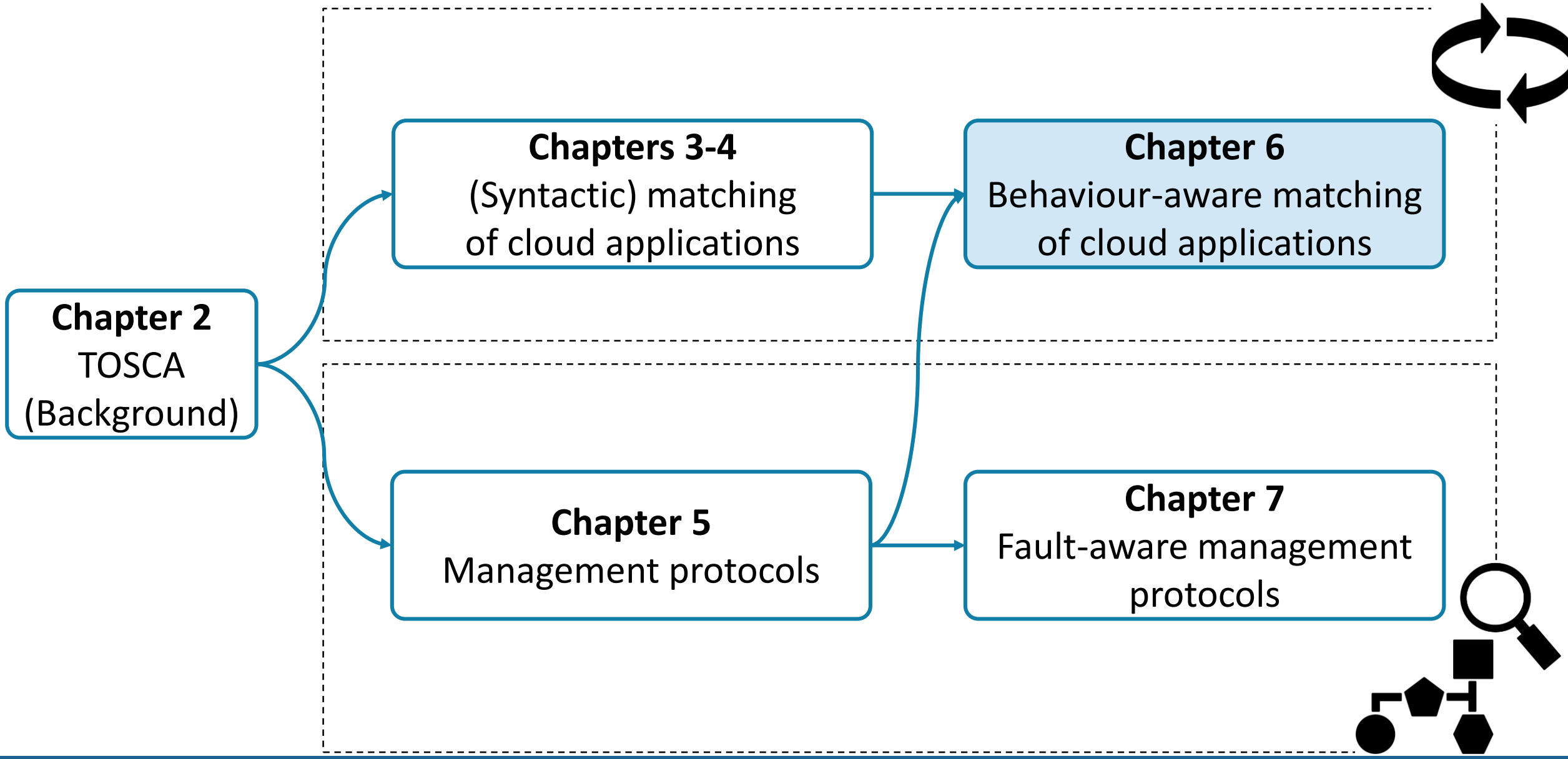





```
INFO  [2016-08-04 14:38:05,071] org.mongodb.driver.cluster: Exception in monitor
 thread while connecting to server unknown:27017
! java.net.UnknownHostException: unknown: unknown error
! at java.net.Inet6AddressImpl.lookupAllHostAddr(Native Method)
! at java.net.InetAddress$2.lookupAllHostAddr(InetAddress.java:928)
! at java.net.InetAddress.getAddressesFromNameService(InetAddress.java:1323)
! at java.net.InetAddress.getAllByName0(InetAddress.java:1276)
! at java.net.InetAddress.getAllByName(InetAddress.java:1192)
! at java.net.InetAddress.getAllByName(InetAddress.java:1126)
! at java.net.InetAddress.getByName(InetAddress.java:1076)
! at com.mongodb.ServerAddress.getSocketAddress(ServerAddress.java:186)
! ... 5 common frames omitted
```

Roadmap

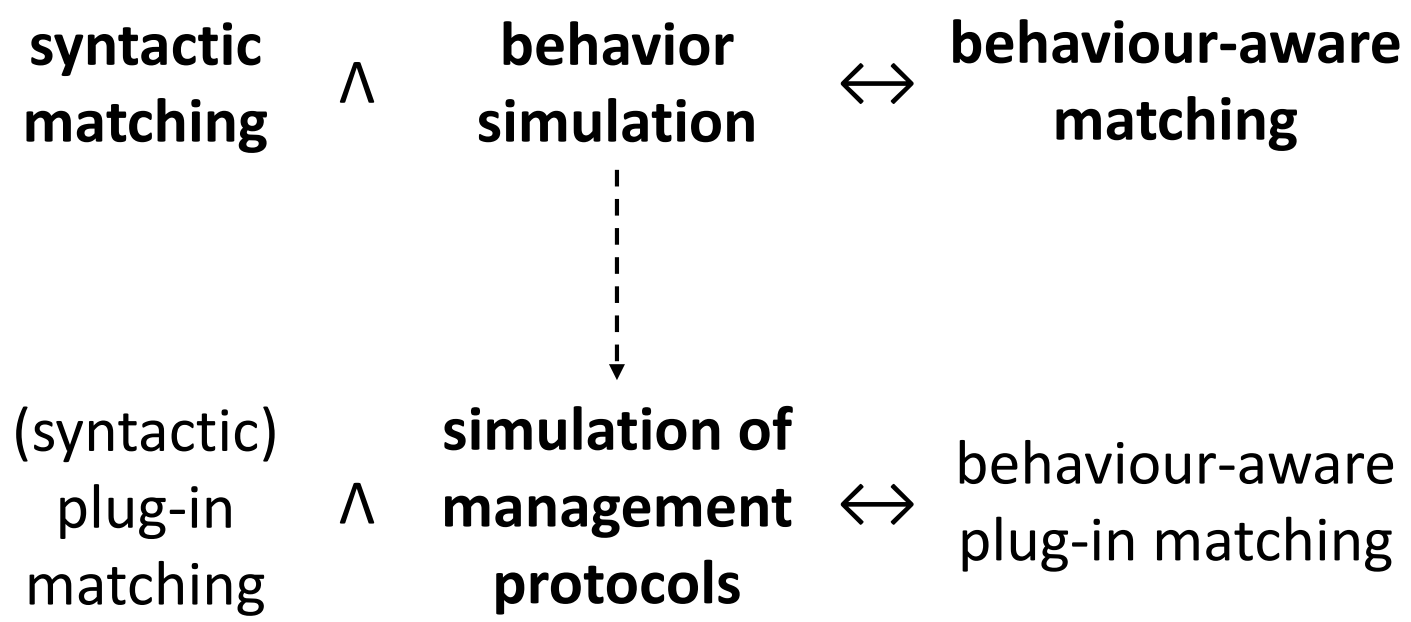
https://tinyurl.com/soldani-thesis



Behaviour-aware matching of cloud applications

We extended the notions of **syntactic matching.**

Idea:



Simulation of management protocols

Two notions of simulation[1] of management protocols:

- **simulation** (for **one-to-one** operation matching)
  M' simulates M iff
  (a) each transition t in M can be simulated by a transition t' in M',
  (b) M' requires less than M, and
  (c) M' offers more than M.

- $\boldsymbol{f}$**-simulation** (for **one-to-many** operation matching)
  M' f-simulates M iff
  (a) each transition t in M can be simulated by the sequence $f(t)$ of transtions in M',
  (b) M' requires less than M, and
  (c) M' offers more than M.

How to **compute** the function $\boldsymbol{f}$?

[1] D. Sangiorgi. **Introduction to bisimulation and coinduction**. Cambridge University Press, 2011.

Computing f-simulations

Algorithm for **finding all functions f** such that M' f-simulates M.

Two steps:

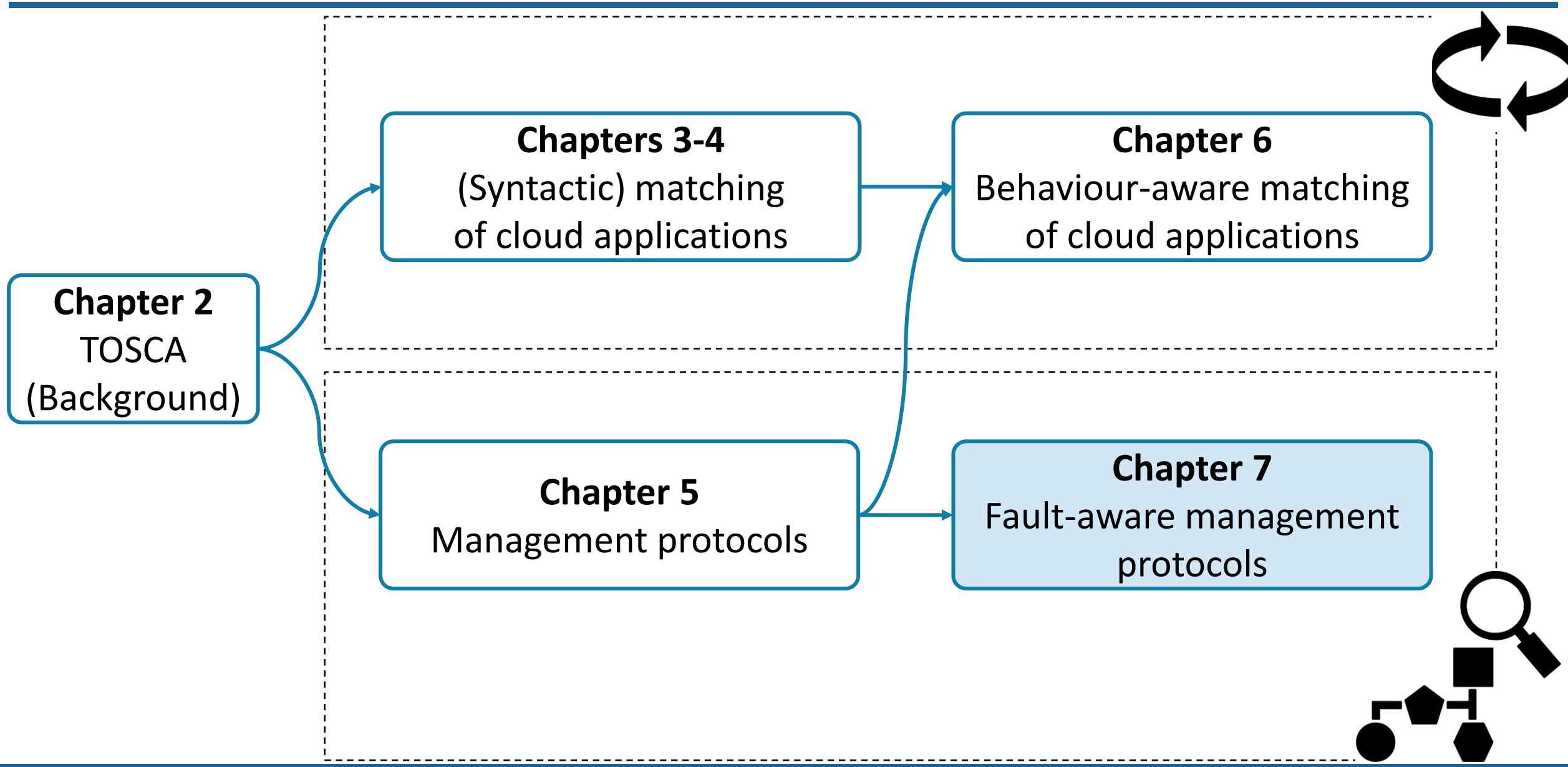
1. **Initialisation**
   - Each transition of M can be simulated by any sequence of transitions in M'.

2. **Iterative refinement**
   - Iteratively remove mappings leading to states that cannot f-simulate.
   - Continue until the mapping cannot be refined any more.

The algorithm is formally proved to be **terminating**, **sound** and **complete**.

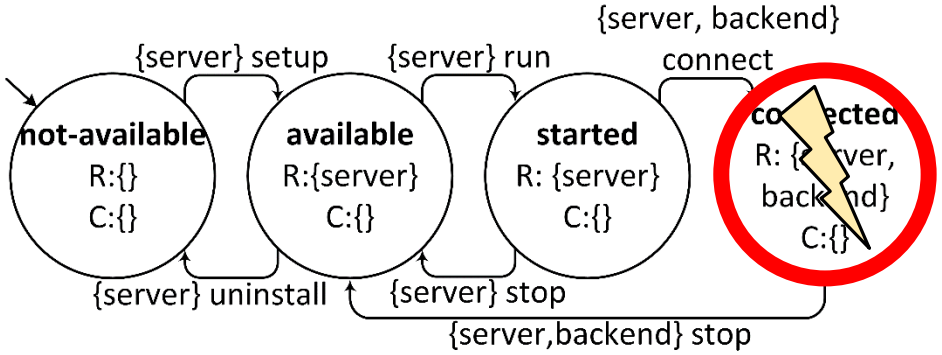
Roadmap

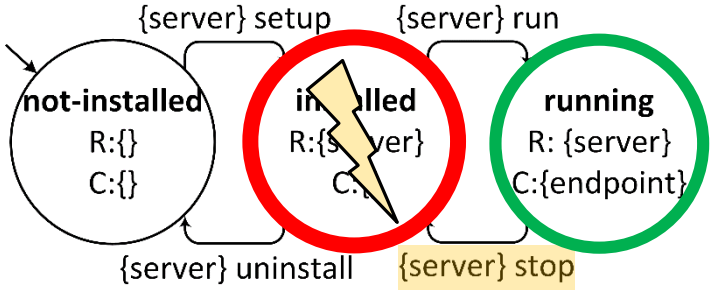
https://tinyurl.com/soldani-thesis

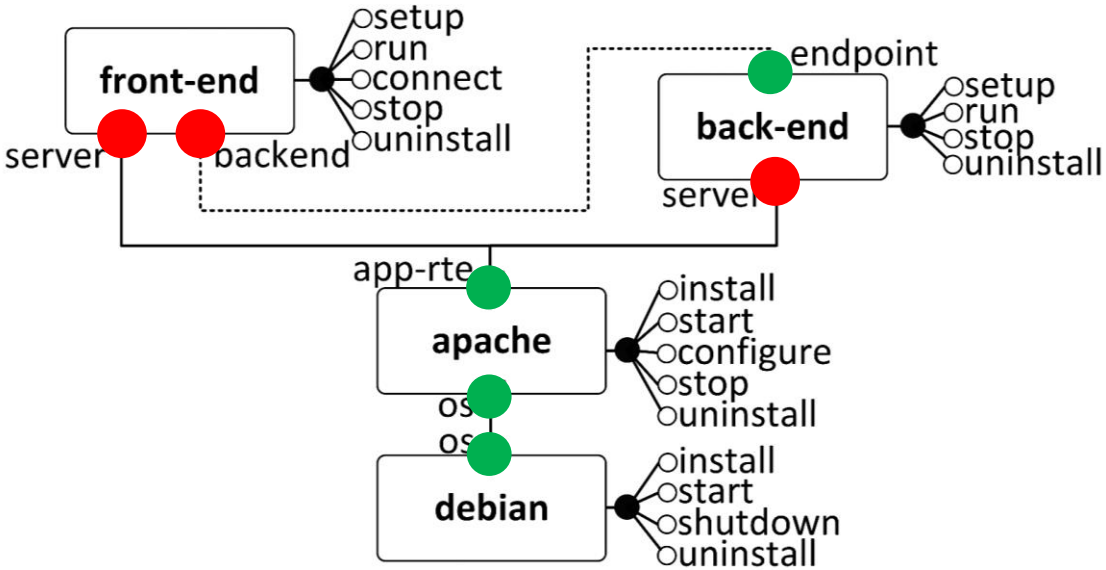


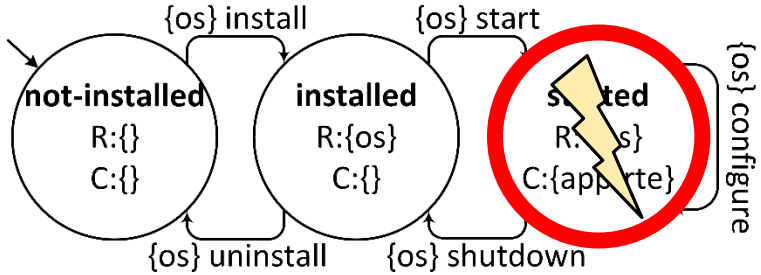
Motivations

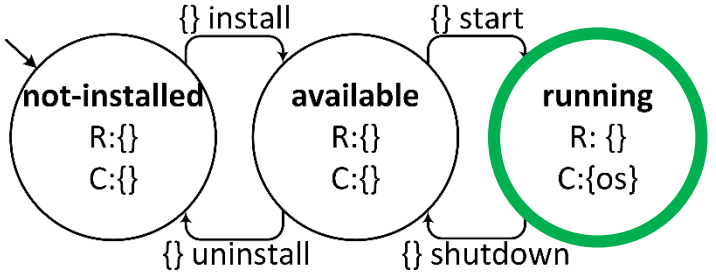
How to handle the **fault of requirements**?











Effects of **misbehaving components**?

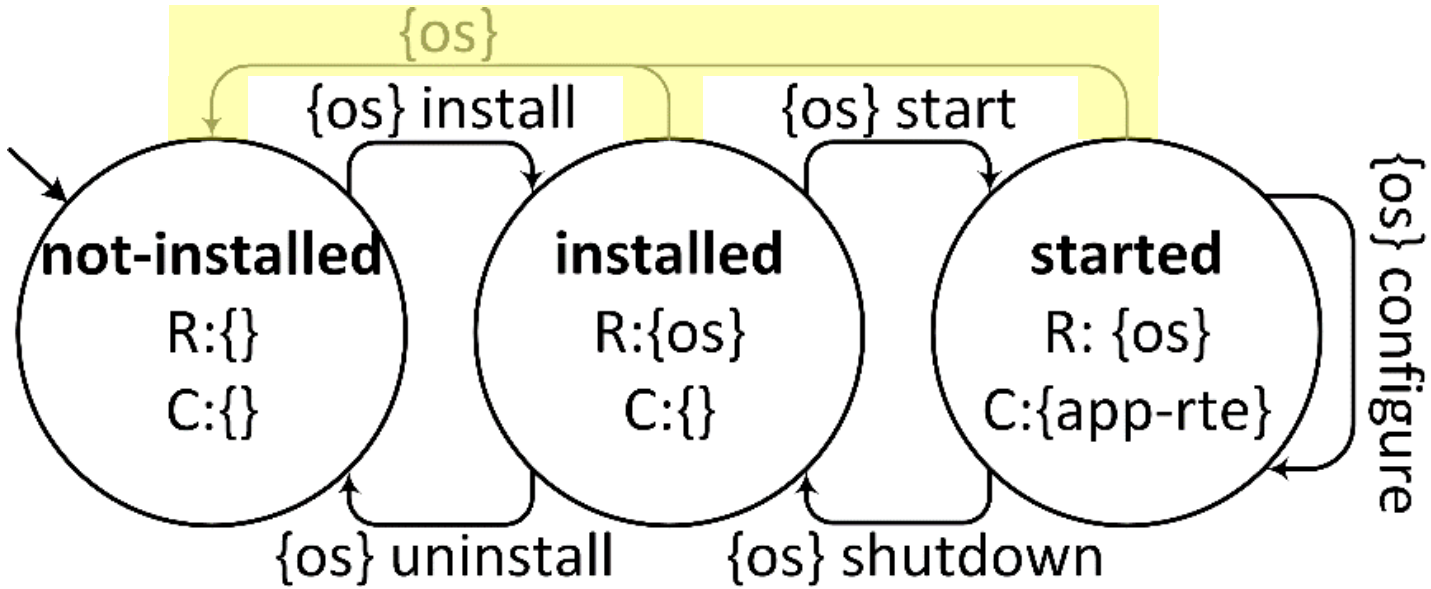
Our approach

Fault-aware management protocols permit
- modelling how nodes behave when faults occurs, and
- analysing/automating application management in presence of faults.

Unexpected behaviour
- naturally modelled in (fault-aware) management protocols
- to permit analysing the (worst possible) effects of a misbehaving component.

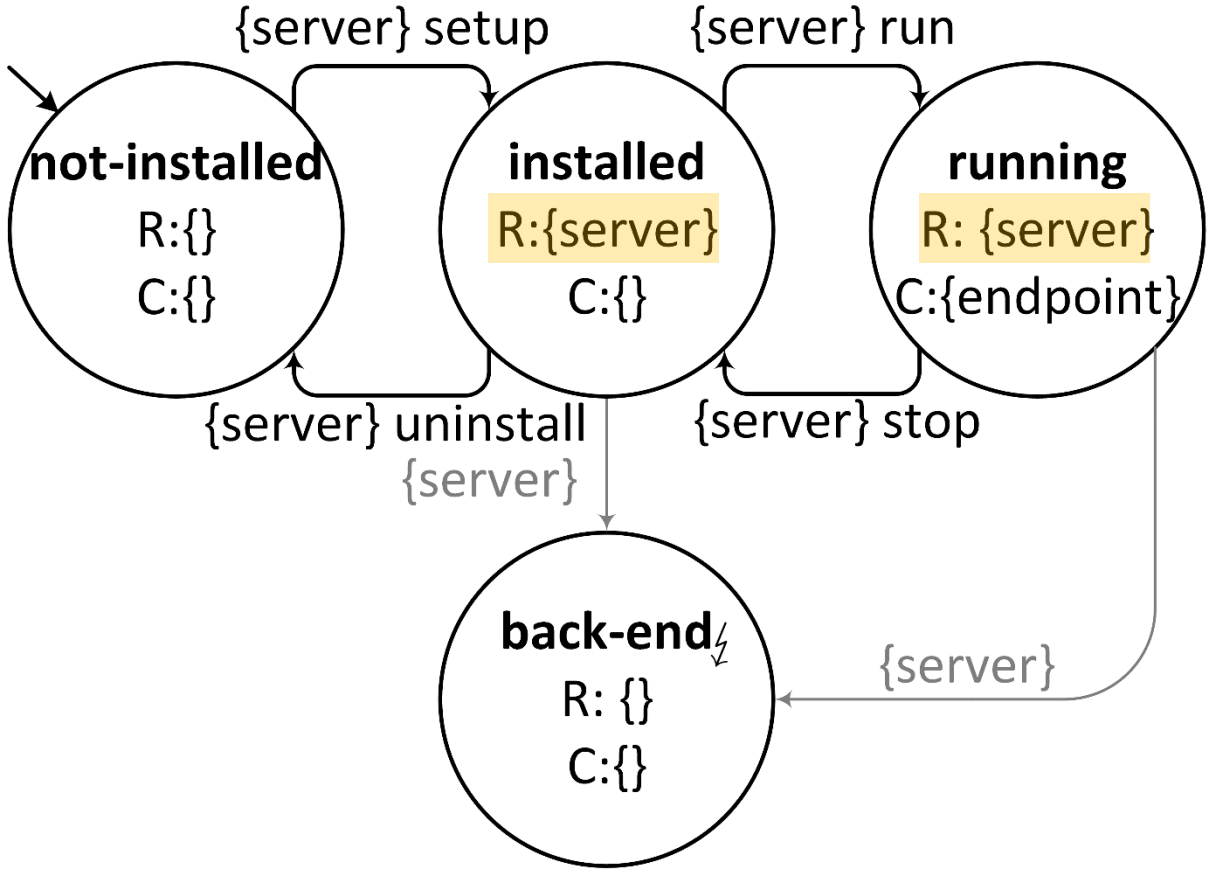
Planning how to **hard recover** applications that are stuck
- since a fault was not properly handled, or
- because of a misbehaving component.

Fault-aware management protocols



Default handling

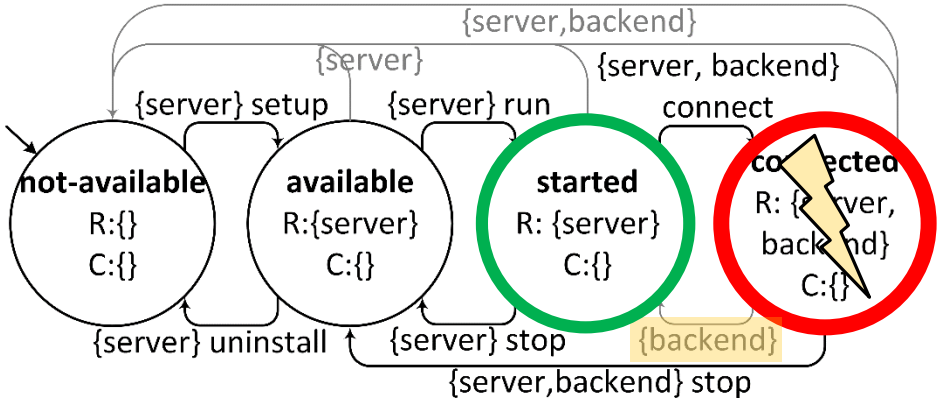
Application designers may leave the handling of some faults unspecified.

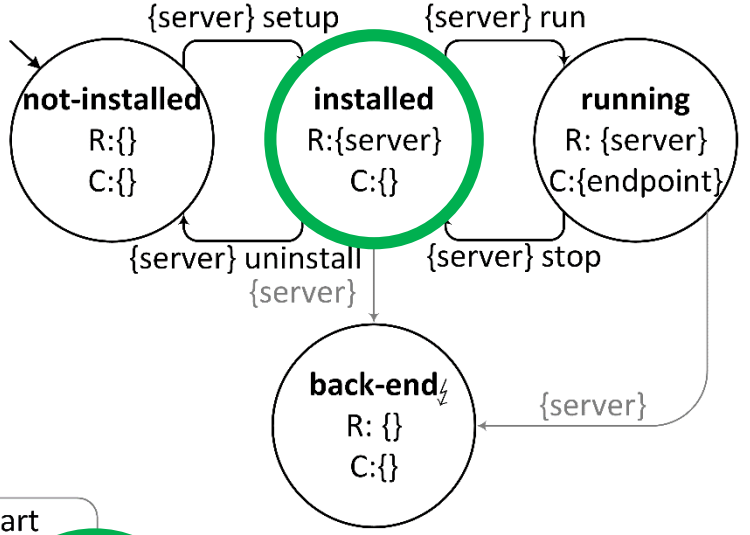


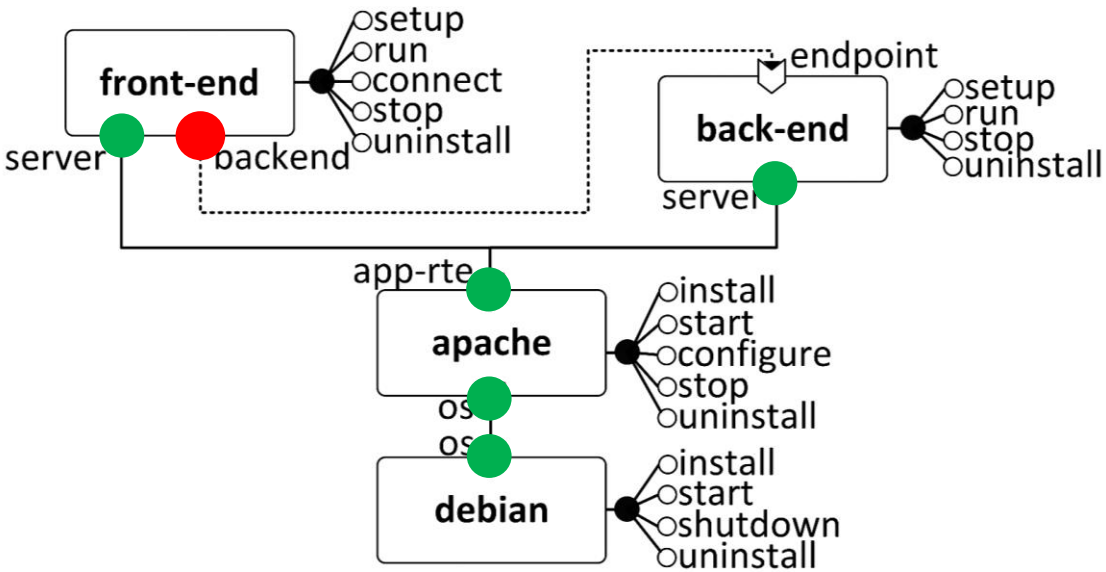
Default handling to a **sink state** that requires/provides nothing (worst-case assumption).

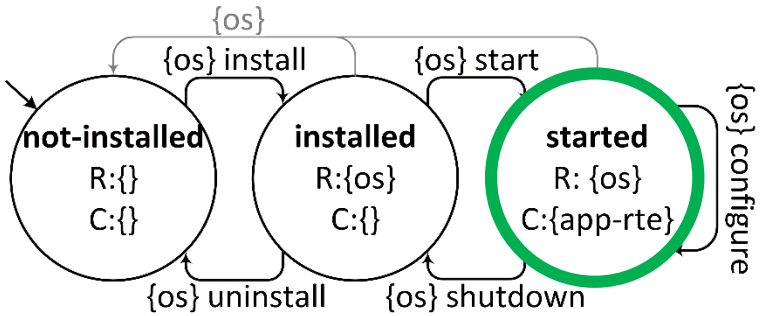
Reasoning with composite applications (and with faults)

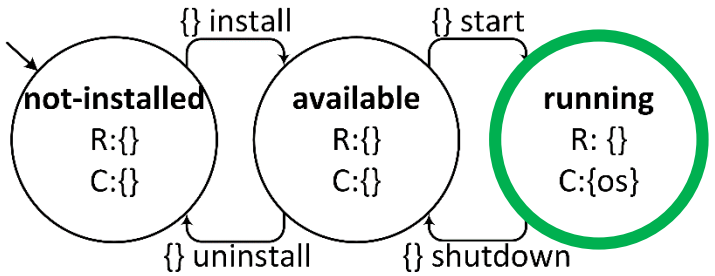
How to handle the **fault of requirements**?











Analysing the management of applications

Validity of plans

» …

Effects of (valid) plans

» …

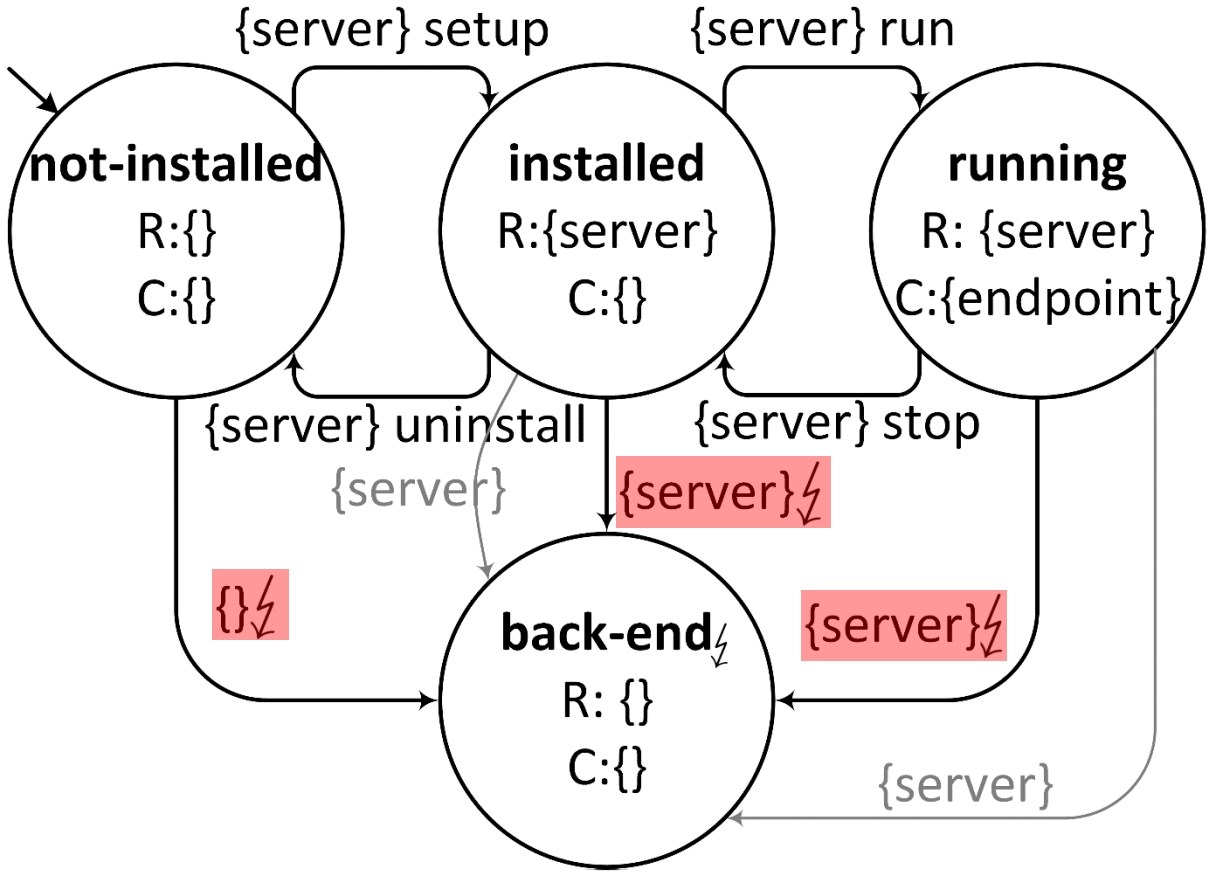
Finding plans (achieving desired goals)

» …

… All previously introduced analyses can still be **automatically performed** (now also taking into account **faults**)

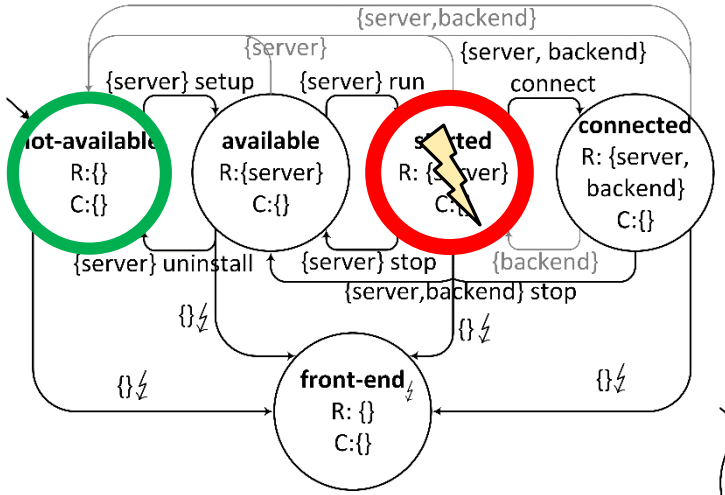
Dealing with «misbehaving components»

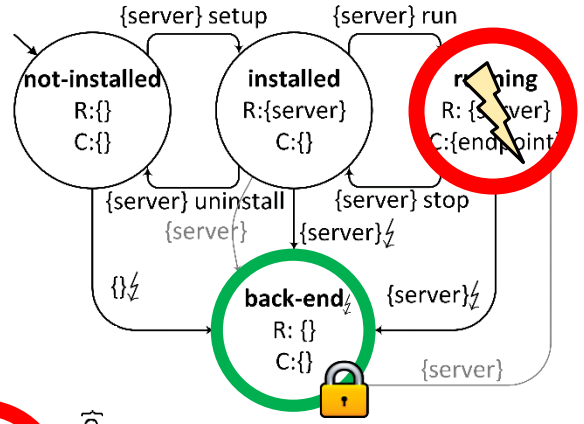
The **unexpected behaviour** of a component can be modelled with a special «**crash**» operation..

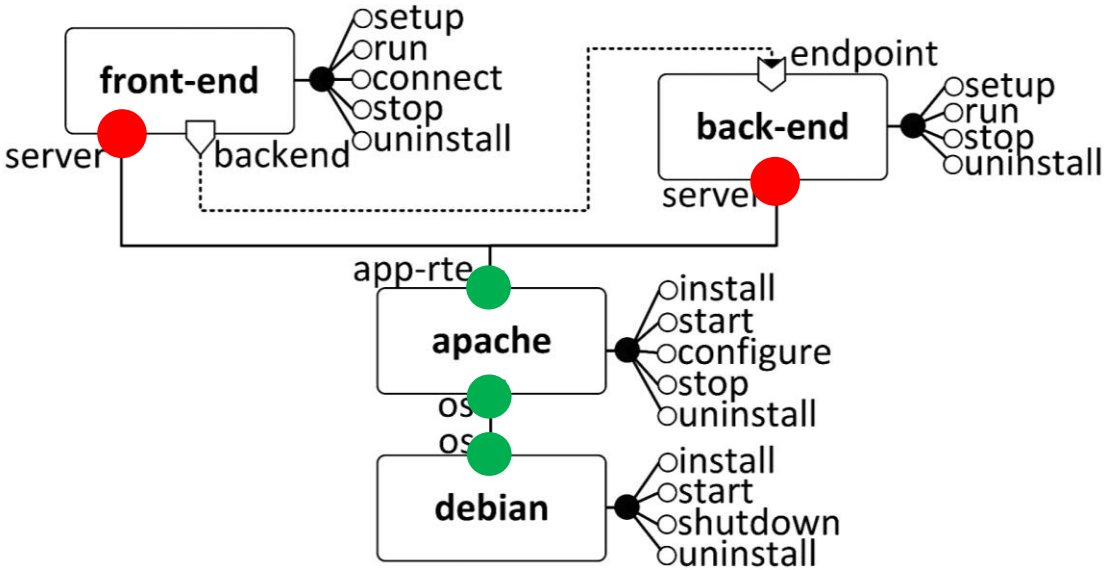


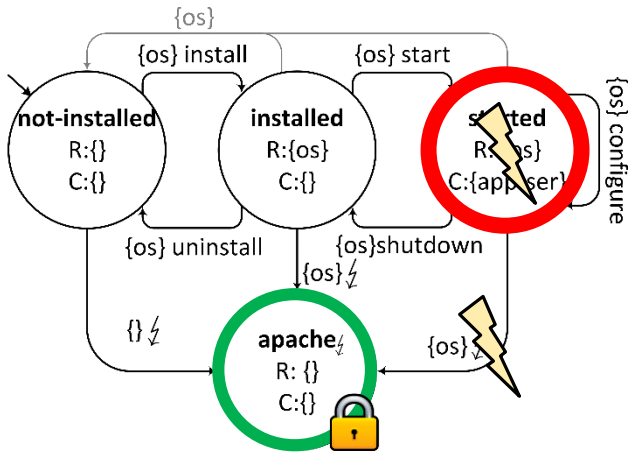
..leading to a **sink state** that provides/requires nothing (worst-case assumption).

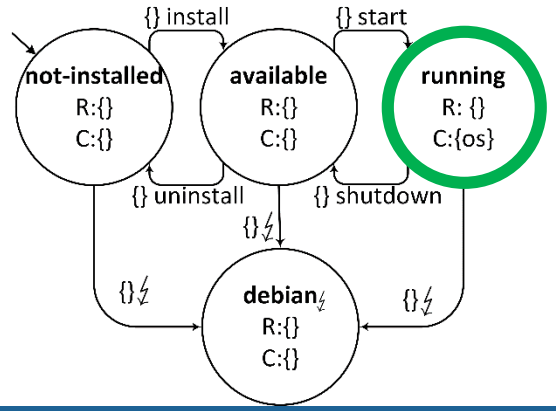
Dealing with misbehaving components







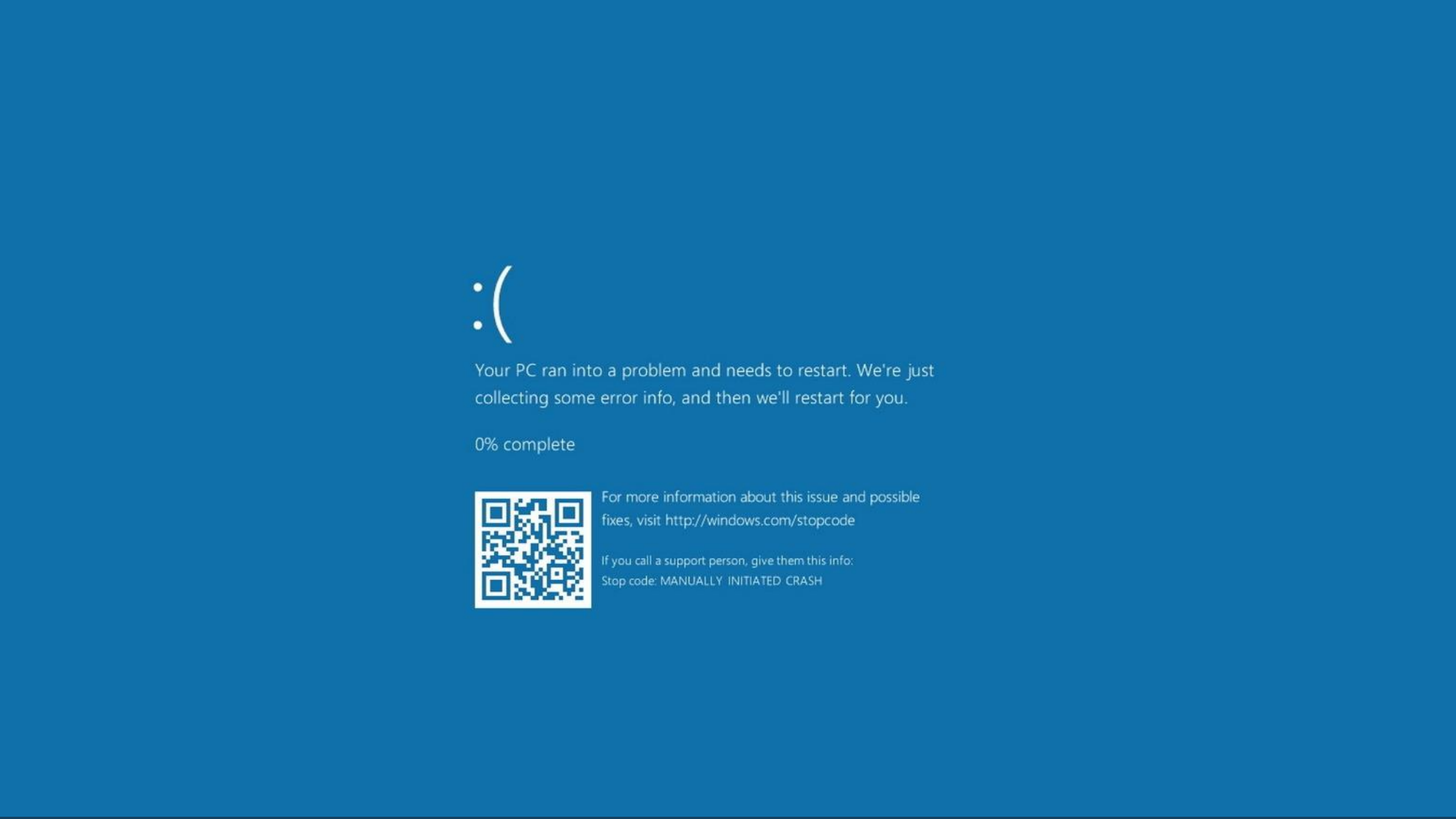




Effects of **misbehaving components**?

Hard recovery

Can **recovery plans** be generated automatically?


:(
Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.
0% complete
For more information about this issue and possible fixes, visit http://windows.com/stopcode
If you call a support person, give them this info:
Stop code: MANUALLY INITIATED CRASH

Hard recovery

Recovery plans can be generated automatically.

Idea (from our experience):

Machine **stuck**, not responding → Forcibly **restart** it — *by **resetting** the whole **system***

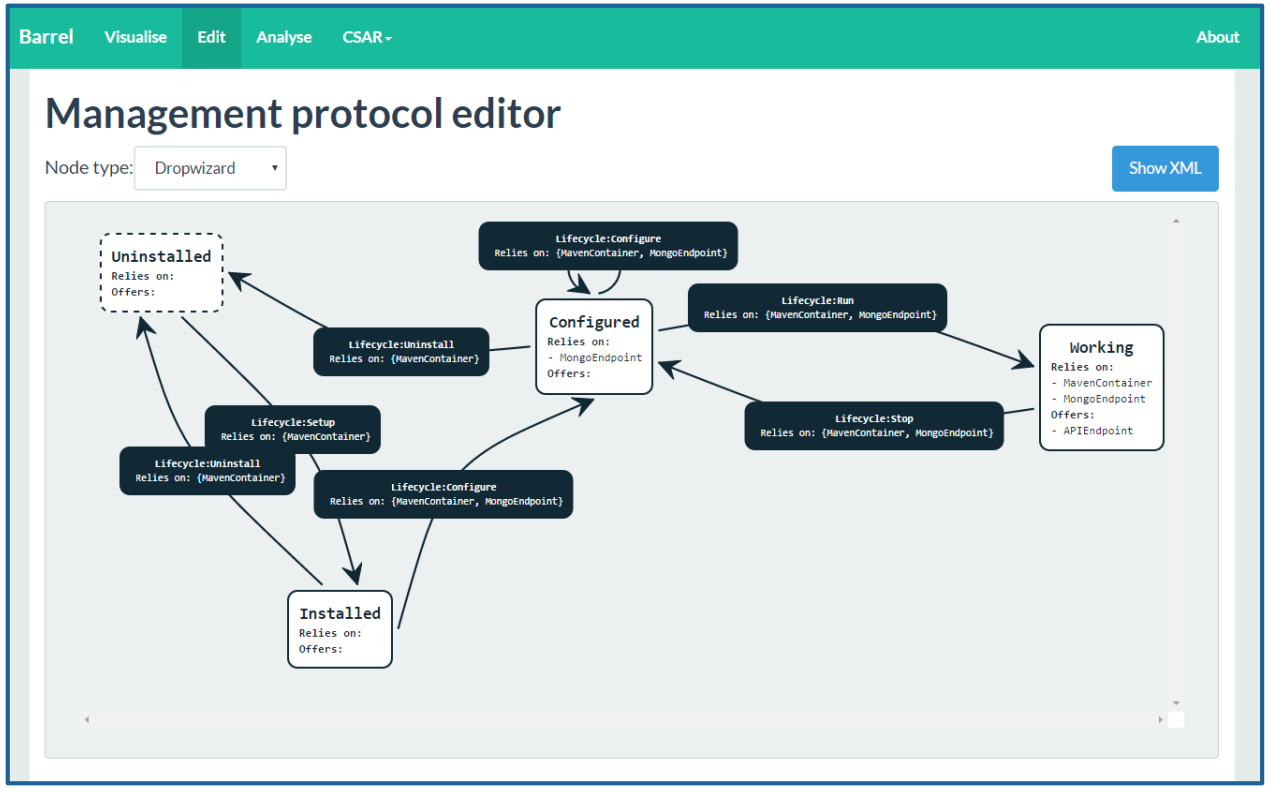
↓ ↓

Node **stuck**, not responding → Forcibly **restart** it — *by **resetting** the **container** node, hence resetting all nodes it contains*

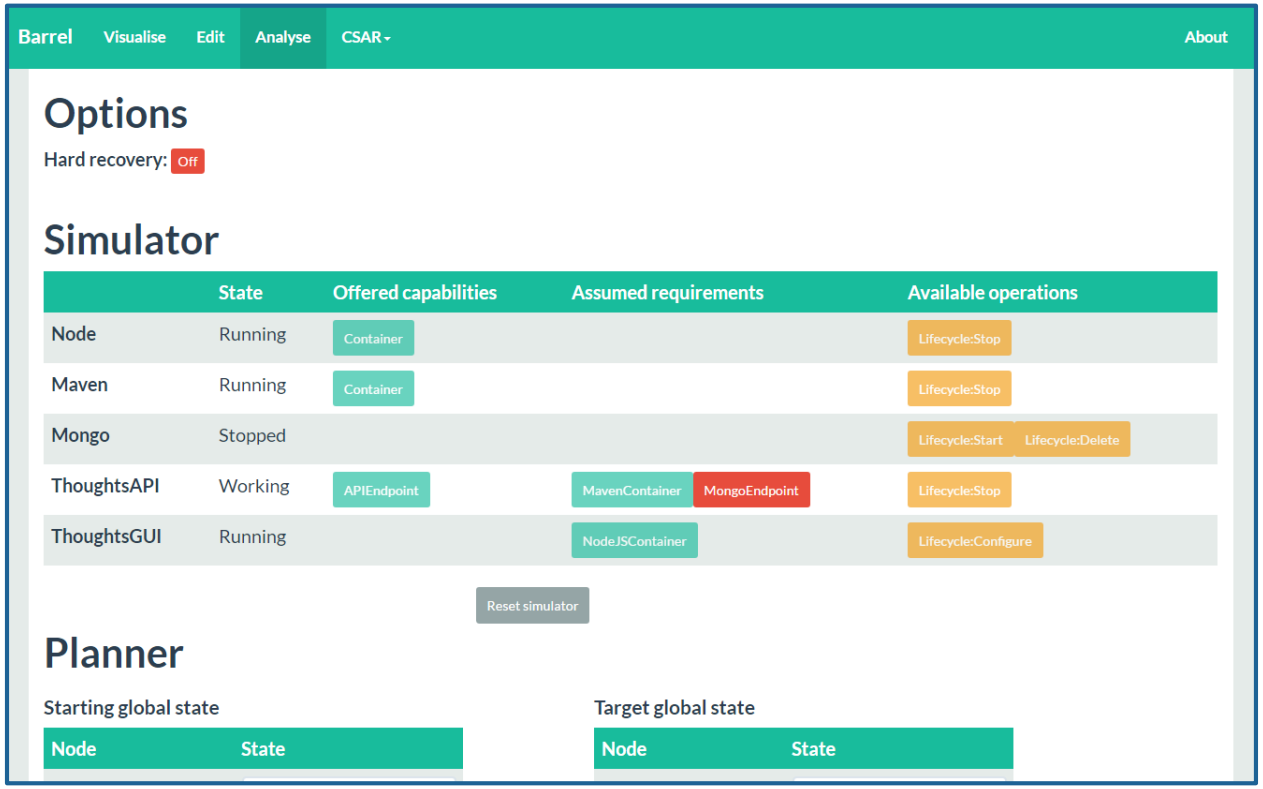
Implementation

Barrel[1]

- Web-based **editor/analyser** of management protocols in TOSCA applications.
- **Open-source** and compatible with the OpenTOSCA ecosystem.



edit



analyse

[1] http://di-unipi-socc.github.io/barrel.

Case study

Thinking

» Real application, made by three components
- **GUI** (deployed on a **NodeJS** Docker cont.)
- **REST API** (deployed on a **Maven** Docker cont.)
- **Mongo** database (running as a Docker cont.)

» **Validation** and **test** of existing deployment plans

» Effects of **misbehaving components**
- e.g., crashed API.

» **Planning**
- e.g., hard recovery of crashed API.

REST API does not return any answer when invoked

Roadmap & publications

https://tinyurl.com/soldani-thesis

A. Brogi, J. Soldani, P. Wang. **TOSCA in a nutshell: Promises and perspectives**. ESOCC, 2014.

Chapter 2
TOSCA
(Background)

A. Brogi, J. Soldani. **Finding available services in TOSCA-compliant clouds**. Sci. Comp. Progr., 2016.

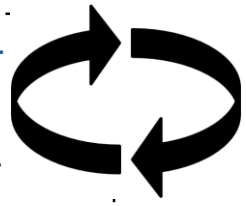
J. Soldani, T. Binz, U. Breitenbücher, F. Leymann, A. Brogi. **TOSCAMART: A Method for Adapting and Reusing Cloud Applications**. JSS, 2016.

Chapters 3-4
(Syntactic) matching
of cloud applications

F. Bonchi, A. Brogi, A. Canciani, J. Soldani. **Behaviour-aware matching of cloud applications**. TASE, 2016.

F. Bonchi, A. Brogi, A. Canciani, J. Soldani. **Simulation-based matching of cloud applications**. Sci. Comp. Progr., 2017.

Chapter 6
Behaviour-aware matching
of cloud applications



Chapter 5
Management protocols

A. Brogi, A. Canciani, J. Soldani. **Modelling and analysing cloud application management**. ESOCC, 2015.

A. Brogi, A. Canciani, J. Soldani, P. Wang. **A Petri net-based approach to model and analyze the management of cloud applications**. Trans. on Petri Nets and other models of Conc., 2016.

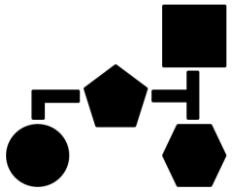
Chapter 7
Fault-aware management
protocols

A. Brogi, A. Canciani, J. Soldani. **Fault-aware application management protocols**. ESOCC, 2016.

A. Brogi, A. Canciani, J. Soldani. **Fault-aware management protocols for composite applications**. JSS, 2018.

Journal
Conference

Conclusions

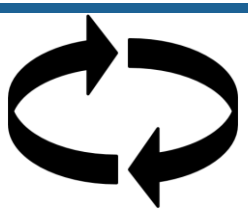


Modelling
composite cloud applications.

Management protocols, which are a **modular**, **compositional**, and **fault-aware** modelling for the management behaviour of application components.

Analysing
composite cloud applications.

Techniques for **analysing** and **automating** the management of composite applications (e.g., validity of plans, effects of plans, planning, hard recovery, etc.).

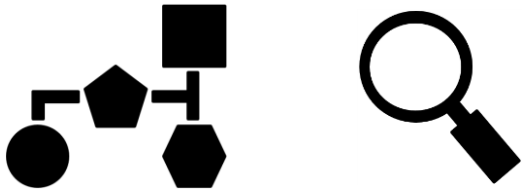
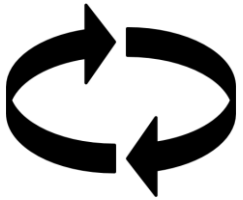


Reusing
composite cloud applications.

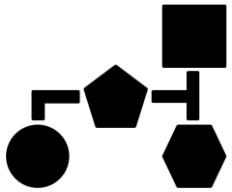
Techniques for **matching and adapting** (fragments of) existing applications, by taking into account both their **structure** and their **behaviour**.

independent from the employed topology model

Conclusions (2)

		
Feasibility assessment	**(a) prototype** implementation **(b) case study**	**(a) prototype** implementations **(b) formal assessment** of all proposed algorithms
Related work	First approach (i) allowing to model and analyse **faults** in composite apps, (ii) dealing with **misbehaving components**, and (iii) allowing to **plan** how to **manage/recover** composite apps.	First approach (i) considering both **functional** and **non-functional** features, and (ii) exploiting **behaviour models/simulation** to go beyond non-relevant operation mismatches.

Future work

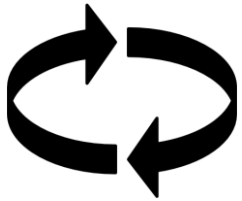


faults generated
during transitions

dynamic reconfiguration
of topologies

cost- and QoS-aware
analyses

management protocols
in TOSCA



full-integration of the
proposed matching techniques

substitutability
assumption

cost- and QoS-aware
matching

Thank you!

Mail: **soldani@di.unipi.it** Web: **http://pages.di.unipi.it/soldani**